LIFTED FIRST-ORDER PROBABILISTIC INFERENCE

BY

RODRIGO DE SALVO BRAZ

B.S., Universidade de São Paulo, 1993
M.S., Universidade de São Paulo, 1998

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2007

Urbana, Illinois

Abstract

There has been a long standing division in AI between logical symbolic and probabilistic reasoning approaches. While probabilistic models can deal well with inherent uncertainty in many real-world domains, they operate on a mostly propositional level. Logic systems, on the other hand, can deal with much richer representations, especially first-order ones. In the last two decades, many probabilistic algorithms accepting first-order specifications have been proposed, but in the inference stage they still operate mostly on a propositional level, where the rich and useful first-order structure is not explicit anymore. In this thesis we present a framework for *lifted* inference on first-order models, that is, inference where the main operations occur on a first-order level, without the need to propositionalize the model. We clearly define the semantics of first-order probabilistic models, present an algorithm (FOVE) that performs lifted inference, and show detailed proofs of its correctness. Furthermore, we describe how to solve the Most Probable Explanation problem with a variant of FOVE, and present a new anytime probabilistic inference algorithm, ABVE, meant to generalize the ability of logical systems to gradually process a model and stop as soon as an answer is available.

To my family, always with me

Acknowledgments

First and foremost, I thank my family for the incredible support they have offered me during my studies. I thank especially my daughter Beatriz, my parents Lúcia and Héber, and co-parent Daniele for support and generosity far greater than what could have been expected of them. I am particularly grateful for the unconditional faith my mother has always placed on me. My gratitude also goes to my siblings Patricia and Marco for their love and understanding.

I am very grateful to my advisor Dan Roth for his unwavering support for all these years, and for the great freedom he allowed me in pursuing my own interests while helping me to put them in perspective. Dan has proved to me that one can be firm, true and unfalteringly nice for years on end. He has also served as a source of inspiration and motivation.

I thank my co-advisor Eyal Amir for his energetic style and willingness to get his hands dirty. Eyal made very substantial contributions and corrections to my research while being a good and fun friend. I also thank my committee members, Gerald DeJong and Adnan Darwiche, who offered valuable feedback, corrections, comments and perspective.

I thank my girlfriend Dafna Shahaf for making my life so much better, not only personally, but also intellectually, always embarking on whatever theorizing I bring up, silly or not. She has served as an incredible source of crunchers, inspiration and energy since we met.

Some friends live far but provided crucial support and encouragement. My friend Sergio Suiama is a constant source of joy in my life, and Maria Williams always someone to turn to for deep understanding.

My life in Urbana has been improved by many good friends. I am especially grateful to Dav Zimak and Samarth Swarup, in their triple role as good friends, colleagues and roommates. Arie Ratinov and Vasin Punyakanok are good friends who have honored me many times with their interest and insightful observations. I am grateful to Mark Sammons for his kindness and constant willingness to revise my texts. Heather Cannon, Joana Maria, Deepak Ramachandran, Apu Kapadia, Prasad Naldurg, Kiran Lakkaraju, Arthur Kantor, Alex Klementiev, Sariel Har-Peled, Geta Sampemane, Cigdem Sengur, Vivek Srikumar, among others, have all brightened my life here.

I am grateful for the support received from Cycorp, the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) program, NSF grant ITR-IIS-0085980, and a Defense Advanced Research Projects Agency (DARPA) grant HR0011-05-1-0040.

Table of Contents

List of Figures

List of Symbols

$LV(\alpha)$	logical variables in object α.
$RV(\alpha)$	random variables defined by object α.
ϕ_g	the potential function of parfactor g.
A_g	the atom parameters of parfactor g.
C_g	the constraint of parfactor g.
$C_{\|L}$	constraints projected to a set of logical variables L.
$g\theta$	parfactor $(\phi_g, A_g\theta, C_g\theta)$ for substitution θ.
$U_L(C_g)$	USCP of L with respect to C_g.
$fs(G)$	the fusion of a set of parfactors G.
$\vec{N}_A$	a multinomial counter on the set of atoms A.
$\#(v, \vec{N}_E)$	defined as $\prod_{i=1}^{k} \vec{N}_{E_i,v_i}$.
G_E	subset of parfactors G which depend on $RV(E)$, for E a set of atoms.
$G_{\neg E}$	subset of parfactors G which do not depend on $RV(E)$.
$\top$	tautology formula.
$p^{\mathbf{P}}$	the potential component of a potential-assignment pair.
$p^{\mathbf{A}}$	the assignment component of a potential-assignment pair.
$\mathbb{P}(\mathbf{X})$	the space of probability distributions on random variables $\mathbf{X}$.
$\Phi_{\mathbf{Q}}$	the subset of potential functions Φ with at least one of the random variables in $\mathbf{Q}$ as an argument.

Chapter 1

Introduction

For decades after the field of Artificial Intelligence (AI) was established, its most prevalent form of representation and inference was logical, or at least symbolic representations that were in a deeper sense equivalent to a fragment of logic. While highly expressive, this type of model lacked a sophisticated treatment of *degrees* of uncertainty, which permeates real-world domains, especially the ones usually associated with intelligence, such as language, perception and common sense reasoning.

In time, probabilistic models became an important part of the field, incorporating probability theory into reasoning and learning AI models. Since the 80s the field has seen a surge of successful solutions involving large amounts of data processed from a probabilistic point of view, applied especially to Natural Language Processing and Pattern Recognition.

This success, however, came with a price. Typically, probabilistic models are less expressive and flexible than logical or symbolic systems. Usually, they involve propositional, rather than first-order representations. When required, more expressive, higher level representations are obtained by ad hoc manipulations of lower level, propositional systems.

Among probabilistic approaches, graphical models such as Bayesian and Markov networks (BNs and MNs respectively) ([Pea88]) are among the most popular. These models are specified by a set of conditional probabilities (for BNs) or factors, also called potential functions (for MNs). Both conditional probabilities and factors are defined over particular subsets of the available random variables, and map assignments of those random variables to positive real numbers (called *potentials* in MNs). For our purposes, it will be helpful to think of graphical models in general and simply consider conditional probabilities as a type of factor.

For example, in an application for document subject classification, one can specify a dependence between the random variables $subject\_apple, word\_mac$ (which indicate that the subject of the document is "apple" and that the word "mac" is present in it) by defining a factor on their assignments. The higher the potential for a given assignment to these random variables, the more likely it will be in the joint distribution defined by the model.

A limitation of expressivity in graphical models arises when the same dependence holds between different subsets of random variables. For example, we might declare the dependence above to hold also between $subject\_microsoft$, $word\_windows$. In traditional graphical models, we must use separate potential functions to do so, even though the dependence is the same. This brings redundancy to the model and possibly wasted computation. It is also an ad hoc mechanism since it does not cover other sets of random variables exhibiting the same dependence (in this case, some other company and product).

The root of this limitation is that graphical models are propositional (random variables can be seen as analogous to propositions in logic), that is, they do not allow quantifiers and parameterization of random variables by objects. A first-order or relational language, on the other hand, does allow for these elements. With such a language, we can specify a potential function that applies, for example, to *all* tuples of random variables obtained by instantiating X and Y in the tuple

$$subject(X), company(X), product(X, Y), word(Y). \tag{1.1}$$

This way we not only cover both cases presented before, but also unforeseen ones, with a single compact specification.

In the last twenty years, many proposals for probabilistic inference algorithms accepting first-order specifications have been presented ([NH95, KDR00, CPQC03a, FGKP99, Poo93, RD04], among many others), most of which based on the theoretic framework of ([Bac90, Hal90]). However, these solutions still perform inference at a mostly propositional level; they typically instantiate potential functions according to the objects relevant to the present query, thus obtaining a regular graphical model on propositional random variables, and then using a regular inference algorithm on this model. In domains with a large number of objects this may be both costly and essentially unnecessary. Suppose we have a medical application about the health of a large population, with a random variable per person indicating whether they are sick with a certain disease, and with a potential function representing the dependence between a person being sick and that person getting hospitalized. To answer the query "what is the probability that *someone* will be hospitalized?", an algorithm that depends on propositionalization will instantiate *a random variable per person*. However this is not necessary since one can calculate the same probability by reasoning about individuals on a general level, simply using the population size, in order to answer that query in a much shorter time. In fact, the latter calculation would not depend on the population size at all.

Naturally, it is possible to reformulate the problem so that it is solved in a more efficient manner. However, this would require manual devising of a process *specific* to the model or query in question. It is

desirable to have an algorithm that can receive a *general* first-order model and *automatically* answer queries like these without computational waste.

Representing and using the structure of probabilistic models is essential to solving them efficiently. It has been shown that, even for propositional models, both exact and approximate inference are intractable [Rot96, DL93]. Using independences in propositional models has proven crucial to managing the cost of inference with them. Using first-order structure is a further step towards solving models which are otherwise intractable.

A first step in this direction was given by [Poo03], which proposes a generalized version of the Variable Elimination algorithm ([ZP94]) that is *lifted*, that is, deals with groups of random variables at a first-order level. The algorithm receives a specification in which parameterized random variables stand for all of their instantiations and then eliminates them in a way that is equivalent to, but much cheaper than, eliminating all their instantiations at once. For the parameterized potential function (1.1), for example, one can eliminate $product(X, Y)$ in a single step that would be equivalent to eliminating all of its instantiations.

The algorithm in [Poo03], however, applies only to certain types of model because it uses a single elimination operation that can only eliminate parameterized random variables containing all parameters present in the potential function (the method can eliminate $product(X, Y)$ from (1.1) but not $company(X)$ because the latter does not contain the parameter Y). As we will see later, Poole's algorithm uses the operation we call *inversion elimination*. In addition to inversion elimination, we have developed further operations (the main ones called *counting elimination* and *partial inversion*) that broaden the applicability of lifted inference to a greater extent ([dSBAR05, dSBAR06]). These operations are combined to form the First-Order Variable Elimination (FOVE) algorithm presented in this chapter. The cases to which lifted inference applies can be roughly summarized as those containing dependencies where the set of parameters of each parameterized random variable are disjoint or, when this is not the case, where there is a set of parameters whose instantiations create independent solvable cases. We specify these conditions in more detail when explaining the operations, and further discuss applicability in section 3.5. When no lifted inference operation applies to a specific part of a model, FOVE can still apply standard propositional methods to that part, assuring completeness and limiting propositional inference to only some parts of the model.

1.1 Thesis Outline and Contributions

The thesis starts by presenting a survey of the literature on First-Order Probabilistic Inference (FOPI) in Chapter 2. This is vast since FOPI is at the convergence of different and important subfields of AI: logical reasoning, deductive databases, graphical models, machine learning, etc. Each field has produced approaches from its own point of view and flavor, and it is not always simple to draw the relationships between distinct works.

From the survey, it becomes clear that lifted inference has received little attention until recently, since most works are based on some form of propositionalization. Chapter 3 presents the main contribution of the thesis, the First-Order Variable Elimination (FOVE) algorithm, which eliminates entire classes of random variables at each step.

The algorithm is presented for calculating marginal probabilities, the most common probabilistic inference task. The modifications needed for performing another common task, calculating the Most Probable Explanation (MPE), is presented in Chapter 4.

One of the current disadvantages of the FOVE algorithm is its need to *shatter* (a form of preprocessing) the entire model in advance before inference can be done. This makes the cost of answering simple queries depending only on a fraction of the model potentially expensive. For the future, it is desirable to develop a version of FOVE that gradually process the model until an acceptable bound on the query is obtained. For now, we have developed such a gradual processing method, called *Anytime Bounded Variable Elimination* (ABVE), in the propositional level, which we intend to eventually generalize to the first-order level. This method is presented in Chapter 5. We then conclude in Chapter 6.

Several finer details of the framework are presented in the appendices, including a complexity analysis in appendix E.

Chapter 2

A Survey of First-Order Probabilistic Inference

In this chapter we give a survey of research on probabilistic models on expressive (usually close to first-order logic) languages. We have roughly divided this research into different stages.

The 1970s and 80s saw great interest in expert systems [RN03, BS84]. As these systems were applied to real-world domains, coping with uncertainty became more desirable, giving rise to the certainty factors approach, which attach numbers (representing degrees of certainty) to rules that get propagated to conclusions during inference.

Certainty factors systems did not have clear semantics, and often produced surprising and nonintuitive results [Pea88]. The search for more clear semantics for rules gave rise, among other things, to approaches such as Bayesian Networks. These however were essentially propositional, and thus had much less expressivity than logic systems.

The search for clear semantics of probabilities in logic systems resulted in works such as [Nil86, Bac90, Hal90], which laid out the basic theoretic principles supporting probabilistic logic. These approaches, however, did not include efficient inference algorithms.

Works aiming at efficient inference algorithms for first-order probabilistic inference (FOPI) can be divided in two groups, which Pearl [Pea88] calls *extensional* and *intensional* systems. In the first one, statements in the language are more procedural in nature, standing for licenses for propagating truth values that have been generalized from true or false to a gray scale of varying degrees of certainty. In the second group languages, statements work as restrictions determining which possible worlds or interpretations are valid ones. They do not directly correspond to computing operations, nor can they typically be taken into account without regard to other rules (that is, inference is not completely modular). Efficient algorithms have to be devised for these languages that preserve their semantics while doing better than considering the entire model at every step.

Among intensional models, we have further divisions regarding the type of algorithm proposed. One group proposes inference rules similar to the ones used in first-order logic inference (for example, modus ponens). A second one computes, in more or less efficient manners, the set of valid interpretations given a

model. A third one uses sampling to answer queries about a model. Finally, a fourth and more prevalent group constructs a (propositional) graphical model (Bayesian or Markov networks, for example) that answers queries, and uses general graphical model inference algorithms for solving them.

We now present these stages in more detail.

2.1 Expert Systems and Certainty Factors

Expert systems are based on rules meant to be applied to existing facts, producing new facts as conclusions [RN03]. Typically, the context is a deterministic one in which facts and rules are assumed to be certain. Uncertainties from real-world applications are dealt with during the modeling stage where necessary (and often heavy-handed) simplifications are performed.

Certainty factors were introduced for the purpose of allowing uncertain rules and facts, making for more direct and accurate modeling. A rule $A \leftarrow B : c_1$, with $c \in [0, 1]$, indicates that we can conclude B with a degree of certainty of $c_1 \times c_2$, if A is known to be true with a degree of certainty c_2. Given a collection of rules and facts, inference is performed by propagating certainties in this fashion. There are also combination rules for the cases when more than one rule provide certainty factors for the same literal.

A paradigmatic application of certainty factors is the system MYCIN [Sho75], an expert system dedicated to diagnosing diseases based on observed symptoms. Clark & McCabe [CM82] describe using Prolog with predicates containing an extra argument representing its certainty and being propagated accordingly. Shapiro [Sha83] describes a Prolog interpreter that does the same but in a way implicit in the interpreter and language, rather than as an extra argument.

One can see that certainty factors have a probabilistic flavor to them, but formally they are not taken to be probabilistic. This is for good reason: should we interpret them as probabilities, the results they provide would be incorrect in many cases. In fact Heckerman [Hec86] and Lucas [Luc01] discuss situations in which certainty factor computations can and cannot be correctly interpreted probabilistically. One reason they cannot is the incorrect treatment of bidirectional inference: two certainty factor rules $A \leftarrow B : c$ and $B : c$ imply nothing about inference from A to B, while $P(A|B)$ and $P(B)$ place constraints on $P(B|A)$. These problems are further discussed in Pearl [Pea88].

2.2 Probabilistic Logic Semantics

The semantic limitations of certainty factors is one of the motivations for defining precise semantics for probabilistic logics, but such investigations date from at least as far back as Carnap [Car50].

One of the most influential AI works in this regard is Nilsson [Nil86] (a similar approach is given by Hailperin [Hai84]). Nilsson establishes a systematic way, given a set of logic sentences and their respective probabilities of being true, of determining the probabilities of other logic sentences of initially unknown probabilities. To be more precise, the method determines *intervals* of probabilities to these novel sentences, because in principle the original set may be consistent with an entire range of point probabilities for them. For example, knowing that A is true with probability 0.2 and B with probability 0.6 means that $A \wedge B$ is true with probability in $[0, 0.2]$, depending on whether A and B are mutually exclusive, or $A \rightarrow B$, or anything in between.

Formally, Nilsson's system is based on the following linear problem:

$$\Pi = VP$$

$$0 \leq \Pi_j \leq 1$$

$$0 \leq P_i \leq 1$$

$$\sum_i P_i = 1$$

where Π is the vector of probabilities of the sentences in the knowledge base, P the vector of probabilities of the possible worlds (consistent with the sentences) and V is a matrix with $V_{ij} = 1$ if sentence j is true in possible world i, and 0 otherwise. The probabilities of sentences in the knowledge base are incorporated as constraints in this system as well, and linear programming techniques can be used to determine the probability of novel sentences. However, as Nilsson points out, the problem becomes intractable even with a modest number of initially given sentences, since all consistent possible worlds need to be defined and this is an intractable problem. Therefore this framework is of mostly theoretical interest.

Placing the probabilities on the possible worlds, as does Nilsson, makes it easy to express subjective probabilities such as "Tweety flies with probability 0.9" (that is, the sum of probabilities of all possible worlds in which Tweety flies is 0.9). However, probabilistic knowledge can also express statistical facts about the domain such as "90% of birds fly" (which says that, in each possible world, 90% of birds fly). Bacchus [Bac90] provides an elaborate probabilistic logic semantics that includes both types of probabilistic knowledge, making it possible to use both statements above, as well as statements meaning "There is a probability of 0.8 that 90% of birds fly." He also discusses the interplay between the two types, namely the question of when it is correct to use the fact that "90% of birds fly" in order to assume that "a randomly chosen bird flies with probability 0.9," a topic that has both formal and philosophical aspects. Halpern [Hal90] elaborates on the axiomatization of Bacchus, taking probabilities to be real numbers (Bacchus did

not), and is often cited as a reference for this semantics with these two types of probabilities. In subsequent work, the subjective type probability has been much more developed and used, and is also the type involved in propositional graphical models.

Fagin, Halpern, Meggido [FHM90] present a logic to reason *about* probabilities, including their addition and multiplication by scalars. Other works discussing the semantics of probabilities on first-order structures are [Fen80, Gai64, GS82].

2.3 Extensional Approaches

Somewhat parallel to the works defining the semantics of probabilistic logic, there has been much work proposing logic reasoning systems incorporating uncertainty in the explicit form of probabilities (as opposed to certainty factors). These systems often stem from the fields of logic programming and deductive databases, and fit into the category described by [Pea88] as *extensional* systems, that is, systems in which rules work as "procedural licenses" for a computation step instead of a constraint on possible probability distributions. Most of these systems operate on a collection of rules or clauses that propagate generalized truth values (typically, a value or interval in $[0, 1]$).

Kiefer and Li [KL88] provide a probabilistic interpretation and a fixpoint semantics to Shapiro [Sha83]. Wüthrich [Wüt95] elaborates on their work, taking into account partial dependencies between clauses. For example, if each of atoms a, b and c has a prior probability of 0.5 and we have two rules $p \leftarrow a \wedge b$ and $p \leftarrow b \wedge c$, Kiefer and Li will assume the rules independent and assign a probability $0.25 + 0.25 - 0.25 * 0.25 = 0.4375$ to p. Wüthrich's system, however, takes into account the fact that b is shared by the clauses and computes instead $0.25 + 0.25 - 0.5^3 = 0.375$ (that is, it avoids double counting of the case where the two rules fire at the same time, which occurs only when the three atoms are true at once).

One of the most influential works within the extensional approach is Ng and Subrahmanian [NS92]. Here, a logic programming system uses generalized truth values in the form of intervals of probabilities. They define probabilistic logic program as sets of *p-clauses* of the form

$$A : \mu \leftarrow F_1 : \mu_1 \wedge \ldots F_n : \mu_n,$$

where A is an atom, $F_1, \ldots, F_n$ are basic formulas (conjunctions or disjunctions) and $\mu, \mu_1, \ldots, \mu_n$ are probability intervals. A clause states that if the probability of each formula F_i is in μ_i, then the probability

of A is in μ. For example, the clause

$$path(X, Y) : [0.8, 0.95] \leftarrow a(X, Z) : [1, 1] \wedge path(Z, Y) : [0.85, 1]$$

states that, if $a(X, Z)$ is certain (probability in interval $[1, 1]$ and therefore 1) and $path(Z, Y)$ has probability in $[0.85, 1]$, then $path(X, Y)$ has probability in $[0.8, 0.95]$. Probabilities of basic formulas F_i are determined from the probability intervals of their conjuncts (disjuncts) by taking into account the possible correlations between them (similarly to what Nilsson does). The authors present a fixpoint semantics where clauses are repeatedly applied and probability intervals successively narrowed up to convergence. They also develop a model theory determining what models (sets of distributions on possible worlds) satisfy a probabilistic logic program, and a refutation procedure for querying a program.

Lakshmanan and Sadri [LS94] proposes a system similar to Ngo and Subrahmanian, while keeping track of both the probability of each atom as well of its negation. Additionally, it uses configurable independence assumptions for different clauses, allowing the user to declare whether two atoms are independent, mutually exclusive, or even the lack of an assumption (as in Nilsson). Lakshmanan [Lak94] separates the qualitative and quantitative aspects of probabilistic logic. Dependencies between atoms are declared in terms of the boolean truth values of a set of *support* atoms. Only later is a distribution assigned to the support atoms, consequently defining distributions on the remaining atoms as well. The main advantage of the approach is the possibility of investigating different total distributions, based on distributions on the support set, without having to recalculate the relationship between atoms and support set. The algorithms works in ways similar to Ngo and Haddawy [NH95] and Lakshmanan and Sadri [LS94], but propagates support set conditions rather than probabilities. Support sets are also a concept very similar to the *hypotheses* used in Probabilistic Abduction by Poole [Poo93] (see next section).

2.4 Intensional Approaches

We now discuss intensional approaches to probabilistic logic languages, where statements (often in the form of rules) are interpreted as restrictions on a globally defined probability distribution. This probability distribution is over all possible worlds or, in other words, on assignments to the set of all possible random variables in the language. Statements typically pose constraints in the form of conditional probabilities, and sometimes also as conditional independence relations (as mentioned in section 2.2, another possibility would be *statistical* constraints, but this has not been explored in any works to our knowledge).

The algorithms in intensional approaches, when available, are arguably more complex than extensional

approaches, since their steps do not directly correspond to the application of rules in the language and need to respect the global distribution while being as local as possible (for efficiency reasons).

We cover four different types of intensional approaches: deduction rules, exhaustive computation of possible worlds, sampling, and Knowledge Based Model Construction (KBMC).

2.4.1 Deduction Rules

Classical logic deduction systems often work by receiving a model specified in a particular language and using deduction rules to derive new statements (guaranteed to be true) from subsets of previous statements. Some work has been devoted to devising similar systems when the language is that of probabilistic logic.

This method is particularly challenging in probabilistic systems because probabilistic inference is not as modular as classical logical inference. For example, while the logical knowledge of $A \Rightarrow B$ allows us to deduce B given that $A \wedge \varphi$ is true for any formula φ, knowing $P(B|A)$ in itself does not tell us *anything* about $P(B|A \wedge \varphi)$. One needs to consider the *global* knowledge when establishing the conditional probability of B. Classical logic reasoning shows a modularity that is harder to achieve in a probabilistic setting.

One way of making probabilistic inference more modular is to use knowledge about conditional independences between random variables. If we know that B is independent of any other random variable given A, then we can calculate $P(B|A \wedge \varphi)$ from $P(B|A)$. This has been the approach of graphical models such as Bayesian and Markov networks [Pea88], where independences are represented in the structure of a graph over the set of random variables (which form its vertices).

The computation steps of specific inference algorithms for graphical models (such as Variable Elimination [ZP94]) could be cast as deduction rules, much like in classical logic. However this is not traditionally done, mostly because inference rules are typically described in a logic-like language and graphical models are not. When dealing with a *first-order* probabilistic logic language, however, this approach becomes more appropriate.

Luckasiewicz [Luk99] uses inference rules for solving trees of probabilistic conditional constraints over basic events. These trees are similar to Bayesian networks, with each node being a random variable and each edge being labeled by a conditional probability table. However, these trees are not meant to encode independence assumptions. Besides, conditional probabilities can also be specified in intervals.

Frisch and Haddawy [FH94] present a set of inference rules for probabilistic propositional logic with interval probabilities. They characterize it as an anytime system since inference rules will increasingly narrow those intervals. They also provide more modular inference by allowing statements on conditional independences of random variables, which are used by certain rules to derive statements based on local

information.

Koller and Halpern [KH96] investigate the use of independence information for FOPI based on inference rules. They use this notion to discuss the issue of *substitution* in probabilistic inference. While substitution is fundamental to classical logic inference, it is not sound in general in probabilistic inference. For example, inferring $P(q(A)) = \frac{1}{3}$ given $\forall P(q(X)) = \frac{1}{3}$ is not sound. Consider three possible worlds w_1, w_2, w_3 containing the three objects o_1, o_2, o_3 each, where $q(o_i)$ is 1 in w_i and 0 otherwise. If each possible world has a probability $\frac{1}{3}$ of being the actual world, then $\forall P(q(X)) = \frac{1}{3}$ holds. However, if A refers to o_i in each w_i, then $P(q(A)) = 1$. While this problem can be solved by requiring constants to be rigid designators (that is, each constant refers to the same object in all worlds), the authors argue that this is too restrictive. Their solution is to use information on independence. They show that when the statements $\forall P(q(X)) = \frac{1}{3}$ and $x = A$ are independent, one can derive $P(q(A)) = \frac{1}{3}$. Finally, they discuss the topic of using statistical probabilities as a basis for subjective ones (the two types discussed by Bacchus [Bac90] and Halpern [Hal90]) based on independences.

2.4.2 Exhaustive Computation of Possible Worlds

Another type of intensional system is the one in which the available algorithms exhaustively compute the set of possible worlds. Therefore, these algorithms are not particularly efficient, although some use dynamic programming techniques for speedup. The point of these works is usually to present a language with a well-defined semantics and algorithm.

Riezler [Rie97] presents a probabilistic account of Constraint Logic Programs (CLPs) [JL87]. In regular logic programming, the only constraint over logical variables are equational constraints coming from unification. CLPs generalize this by allowing other constraints to be stated over those variables. These constraints are managed by special-purpose constraint solvers as the derivation proceeds, and the failure in satisfying a constraint determines the failure of the derivation. Probabilistic Constraint Logic Programs (PCLPs) are a stochastic generalization of CLPs, where clauses are annotated with a probability and chosen for the expansion of a literal according to that probability, among the available clauses with matching heads. The probability of a derivation is determined by the product of probabilities associated to the stochastic choices. In fact, PCLPs are a generalization of Stochastic Context-Free Grammars (SCFGs) [LY90], the difference between them being that PCLP symbols have arguments in the form of logical variables with associated constraints while grammar symbols do not. For this reason, PCLP derivations can fail while SCFGs will always succeed. This presents a complication for PCLP algorithms because the probability has to be normalized with respect to the sum of *successful* derivations only. It also makes the use of efficient dynamic

programming techniques such as the inside-outside algorithm [Bak79] not adequate for PCLPs, forcing us to compute all possible derivations of a query, which amounts to calculating the set of possible worlds satisfying it. Riezler focuses on presenting an algorithm for learning the parameters of a PCLP from incomplete data, in what is a generalization of the Baum-Welch algorithm for HMMs [Bau72].

Stochastic Logic Programs [Mug95b, Cus99] are very similar to PCLPs, restricting themselves to regular logic programming (e.g., Prolog). This line of work is more focused on the development of an actual system on top of a Prolog interpreter and used with Inductive Logic Programming techniques such as Progol [Mug95a]. Like Riezler, in [Cus99] Cussens develops methods for learning parameters using Improved Interative Scaling [DPDPL97] and the EM algorithm [DLR77].

Luckasiewicz [Luk98] presents a form of Probabilistic Logic Programming that complements Nilsson's [Nil86] approach. Nilsson considers all possible worlds consistent with the given knowledge and builds a linear program in order to assign probabilities to sentences. Luckasiewicz essentially does the same by using logic programming for both determining the set of possible worlds and the linear program. Therefore, his approach suffers from the same intractability issues faced by Nilsson's.

Baral et al. [BGR04] use answer set logic programming to implement a powerful probabilistic logic language. Its distinguishing feature is the possibility of specifying observations and actions, with their corresponding implications with respect to causality, as studied by Pearl [Pea00]. However, the implementation, using answer set Prolog, depends on determining all possible consistent worlds.

2.4.3 Sampling Approaches

Because building all possible worlds given a program is very expensive, approximation solutions become an attractive alternative.

Sato [SK97] presents PRISM, a full-featured Prolog interpreter extended with probabilistic switches that can be used to encode probabilistic rules and facts. These switches are special built-in predicates that randomly succeed or not, following a specific probability distribution. They can be places in the body of a clause so that that clause will succeed or not following that distribution (when the rest of the body succeeds). Therefore, multiple executions of the program will yield different random results that can be used as samples. A query can then be answered by multiple executions which sample its possible outcomes. Sato also provides a way of learning the parameters of the switches by using a form of the EM algorithm [SK00].

BLOG [MMR+05] is a first-order language that, although not following the usual logic programming notation, has similar expressiveness. It is similar in form to BUGS [STBG94], a specification language for propositional generative models. The main distinction of BLOG is its open world assumption; it does not

require that the number of objects in the world be set a priori, using instead a prior on this number and also keeping track of identities of objects with different names. BLOG computes queries by sampling over possible worlds.

2.4.4 Knowledge Based Model Construction

We now present the most prominent family of models in the field of FOPI models, Knowledge Based Model Construction (KBMC). These approaches work by generating a propositional graphical model from a first-order language specification that answers the query at hand. This construction is usually done in a way specific to this query, ruling out irrelevant portions of the graph so as to increase efficiency.

Many KBMC approaches use a first-order logic-like specification language, but some use different languages such as frame systems, parameterized fragments of Bayesian networks, and description logics. Some build Bayesian networks while others prefer Markov networks (and in one case, Dependency Networks [HCM+00]).

While KBMC approaches try to prune sections of underlying graphical models which are irrelevant to the current query, there is still potentially much wasted computation because it may replicate portions of the graph which require essentially identical computations. For example, a problem may involve many employees in a company, and the underlying graphical model will contain a distinct section with its own set of random variables for each of them (representing their properties), even though these sections all have essentially the same structure. Often the same computation step will be applied to each of those sections, while it is possible to perform it only once in a generalized form. Avoiding this waste is the object of Lifted First-Order Probabilistic Inference [dSBAR05, dSBAR06], presented in chapter 3.

The most commonly referenced KBMC approach is that of Breese [Bre91, WBG92], although Horsch and Poole [HP90] had presented a similar solution a year before. [Bre91] defines a probabilistic logic programming language, with Horn clauses annotated by probabilistic dependencies between the clause's head and body. Once a query is presented, clauses are applied to it in order to determine the probabilistic dependencies relevant to it. These dependencies are then used to form a Bayesian network. Backward inference will generate the causal portion of the network relative to the query; forward inference creates the diagnostic part. The construction algorithm uses the evidence in order to decide when to stop expanding the network – there is no need to generate portions that are d-separated from the query by the evidence. In fact, this work covers not only Bayesian networks, but influence diagrams as well, including decision and utility value nodes.

There are many works similar in spirit to [Bre91] and differing only in some details; for example, the

already mentioned Horsch and Poole [HP90], which also uses mostly Horn clauses (it does allow for universal and existential quantifiers over the entire clause body though) as the first-order language. One distinction in this work, however, is the more explicit treatment of the issue of *combination functions*, used to combine distributions coming from distinct clauses with the same head. One example of a combination function is noisy-or [Pea88], which assumes that the probability provided by a single clause is the probability of it making the consequent true regardless of the other clauses. Suppose we have clauses $A \leftarrow B$ and $A \leftarrow C$ in the knowledge base, the first one dictating a probability 0.8 for A when B is true and the second one dictating a probability 0.7 for A when C is true. Then the combination function noisy-or builds a Conditional Probability Table (CPT) with B and C as parents of A, with entries $P(A|B,C) = \{1 - 0.2 \times 0.3, 1 - 0.8 \times 0.3, 1 - 0.2 \times 0.7, 1 - 0.8 \times 0.7\} = \{0.94, 0.76, 0.86, 0.47\}$ for $\{(B = \top, C = \top), (B = \bot, C = \top), (B = \top, C = \bot), (B = \bot, C = \bot)\}$, respectively.

Charniak and Goldman [GC93] expand a deductive database and truth maintenance system (TMS) in order to define a language for constructing Bayesian networks. The Bayesian networks come from the data-dependency network maintained by the TMS system, which is annotated with probabilities. There is also a notion of combination functions. The authors choose not to expand logical languages, justifying this choice by arguing that logic and probability do not correspond perfectly, the first being based on implication while the second, on conditioning.

Poole [Poo93] defines Probabilistic Abduction, a probabilistic logic language aimed at performing abduction reasoning. Probabilities are defined only for a set of predicates, called *hypotheses* (which is reminiscent of the support set in [Lak94]), while the clauses themselves are deterministic. When a problem has naturally dependent hypotheses, one can redefine them as regular predicates and invent a new hypothesis to explain that dependence. While deterministic clauses can seem too restrictive, one can always get the effect of probabilistic rules by using hypotheses as a condition of the rule (like switches in Sato's PRISM [SK97]). The language also assumes that the bodies of clauses with the same head are mutually exclusive, and again this is not as restrictive as it might seem since clauses with non-mutually exclusive bodies can be rewritten as a different set of clauses satisfying this. As in other works in this section, the actual computation of probabilities is based on the construction of a Bayesian network. In [Poo97], Poole extends Probabilistic Abduction for decision theory, including both utility and decision variables, as well as negation as failure.

Glesner and Koller [GK95] presents a Prolog-like language that allows the declaration of facts about a Bayesian network to be constructed by the inference process. The computing mechanisms of Prolog are used to define the CPTs as well, so they are not restricted to tables, but can be computed on the fly. This allows CPTs to be defined as decision trees, for example, which provides a means of doing automatic pruning of

the resulting Bayesian network – if the evidence provides information enough to make a CPT decision at a certain tree node, the descendants of that node, along with parts of the network relevant to those descendants only, do not need to be considered or built. The authors focus on flexible dynamic Bayesian networks that do not necessarily have the same structure at every time slice.

Haddawy [Had94] presents a language and construction method very similar to [HP90, Bre91]. However, he focuses on defining the semantics of the first-order probabilistic logic language directly, and independently of the Bayesian network construction, and proceeds to use it to prove the correctness of the construction method. Breese had done something similar by defining the semantics of the knowledge base as an abstract Bayesian network which does not usually get built itself in the presence of evidence, and by showing that the Bayesian network actually built will give the same result as the abstract one.

Koller and Pfeffer [KP97] present an algorithm for learning the probabilities of noisy first-order rules used for KBMC. They use the EM algorithm applied to the Bayesian networks generated by the model, using incomplete data. This works in the same way as the regular Bayesian network parameter learning with EM, with the difference that many of the parameters in the generated networks are in fact *instances* of the same parameter in a first-order rule. Therefore, all updates on these parameters must be accumulated in the original parameter.

Jaeger [Jae97] defines a language for specifying a Bayesian network whose nodes are the extensions of first-order predicates. In other words, each node is the assignment to the set *all* atoms of a certain predicate. Needless to say, such a network would cause very inefficient inference due to the extremely large number of values for each node. However, it has the advantage of making the semantics of the language very clear (it is just the usual propositional Bayesian network semantics – the extension of a predicate is just a propositional variable with a very large number of values). The author proposes, like other approaches here, to build a regular Bayesian network (with a random variable per ground atom) for the purpose of answering specific queries. He also presents a sophisticated scheme for combination functions, including their nesting.

Koller et al. [GFKP01, KP98] define Probabilistic Relational Models (PRMs), a sharp depart from the logical-probabilistic models that had been proposed until then as solutions for FOPI models. Instead of adding probabilities to some logic-like language, the authors use the formalism of Frame Systems [Min95] as a starting point. The language of frames, similar also to relational databases, is less expressive than first-order logic, which is to the authors one of its main advantages since first-order logic inference is known to be intractable (which only gets worse when probabilities are added to the mix). By using a language that limits its expressivity to what is most needed in practical applications, one hopes to obtain more tractable inference, an argument commonly held in the Knowledge Representation community [LB87]. In fact, Pfeffer

and Koller had already investigated adding probabilities to restricted languages in [KLP97]. In that case, the language in question was that of description logics.

The language of Frame Systems consists of defining a set of objects described by *attributes* – binary predicates relating an object to a simple scalar value – or *relations* – binary predicates relating an object to another (or even self) object. PRMs add probabilities to frame systems by establishing distributions on attributes conditioned on other attributes (in the same object, or related object). In order to avoid declaring these dependencies for each object, this is done at a *scheme* level where classes, or template objects, stand for all instances of a class. This scheme describes the attributes of classes and the relations between them. Conditional probabilities are defined for attributes and can name the conditioning attributes via the relations needed to reach them.

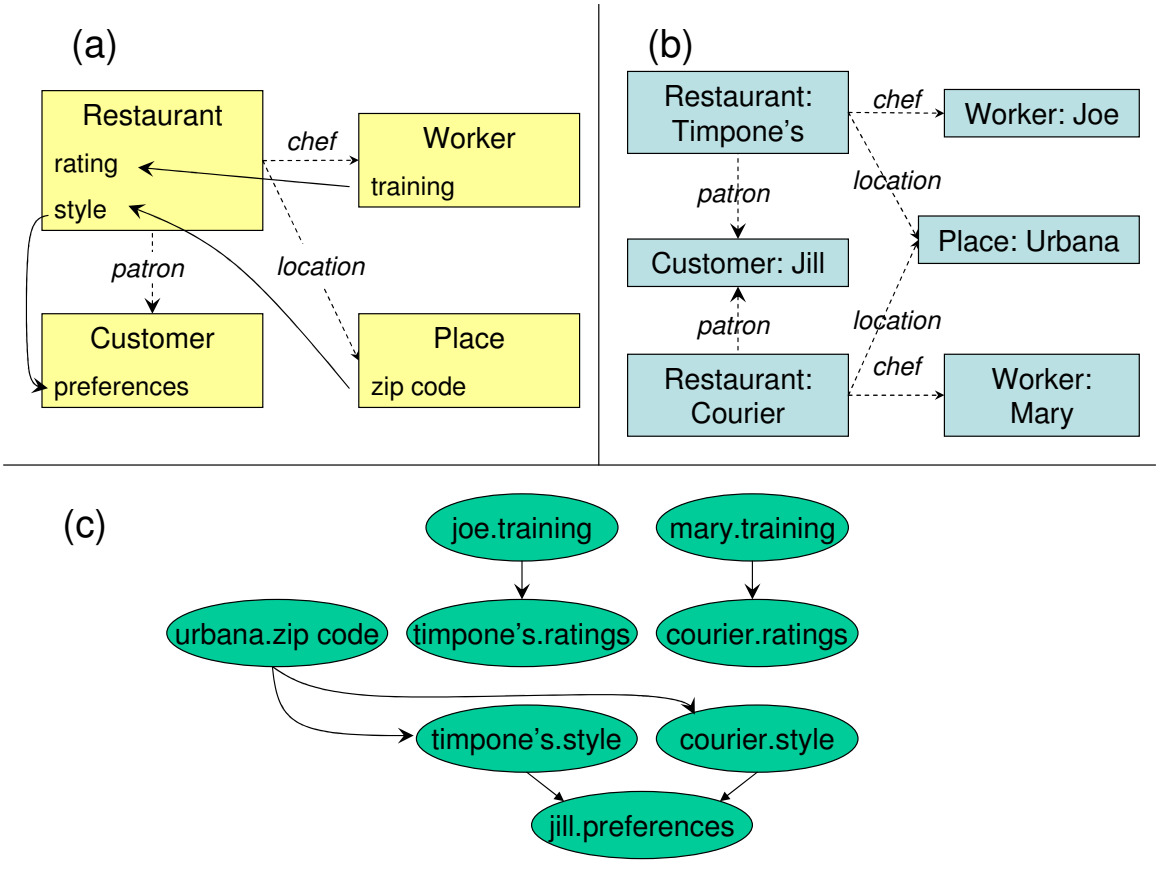

Figure 2.1: (a) A PRM scheme showing classes of objects (rectangles), probabilistic dependencies between their attributes (full arrows) and relationships (dashed arrows). (b) A database skeleton showing a collection of objects, their classes and relationships. (c) The corresponding generated Bayesian network.

As in the previous approaches, queries to PRMs are computed by generating an underlying Bayesian network. Given a collection of objects (a database *skeleton*) and the relationships between them, a Bayesian network is built with a random variable for each attribute in each object. The parents of these random variables in the network are the ones determined by the relations in the particular database, and the CPT filled with the values specified at the template level. An example of this process is show in fig. 2.1.

Note that the set of ancestors of attributes in the underlying network is determined by the relations from one object to another. One could imagine an attribute *rating* of an object representing a restaurant that depends on the attribute *training* of the object representing its chef (related to it by the relationship $chef$. In approaches following first-order representations, $chef$ would be a binary predicate, and each of its instances a random variable. As a result, the ancestors of *rating* would be the attributes *training* of all objects potentially linked to the restaurant by the relationship $chef$, plus the random variables standing for possible pairs in the relationship *cook* itself, resulting in a large (and thus expensive) CPT. PRMs avoid this when they take data with a defined structure where the assignment to relations such as *cook* is know; in this case, the random variables in the relationship $chef$ would not even be included in the network, and the attribute *rating* of each object would have a single ancestor. When relationships are not fixed in advance, we have *structural uncertainty*, which was addressed by the authors in [Get00]. These papers have presented PRM learning of both parameters and structure (that is, the learning of the template level).

PRMs generate Bayesian networks, a directed graphical model that brings a notion of causality. In relational domains it is often then case that random variables depend on each other without a clear notion of causality. Take for example a network of people linked by friendship relationships, with the attribute *smoker* for each person. We might want to state the first-order causal relationship $P(smoker(X)|friends(X,Y), smoker(Y))$ in such a model, but it would create cycles in the underlying Bayesian network (between each pair of *smoker* attributes such as $smoker(john)$ and $smoker(mary)$). For this reason, Relational Markov Networks (RMNs) [TAK02] recast PRMs so they generate undirected graphical models (Markov networks) instead of Bayesian networks. In RMNs, dependencies are stated as first-order features that get instantiated into potential function on cliques of random variables, without a notion of causality or conditional probabilities. The disadvantage of it, however, is that learning in undirected graphical models is harder than in directed ones, involving a full inference step at each expectation step of the EM algorithm.

Relational Dependency Networks (RDNs) [Nev06] provide yet another alternative to this problem. They are the first-order version of Dependency Networks (DNs) (Heckerman, [HCM$^+$00]), which use conditional probabilities but do not require acyclicity. Using directed conditional probabilities avoids the expensive learning of undirected models. However, DNs have the downside of conditional probabilities being no longer

guaranteed consistent with the joint probability defined by their normalized product. Heckerman shows that, as the amount of training data increases, conditional probabilities in a DN will asymptotically converge to consistency. RDNs are sets of first-order conditional probabilities which are used to generate an underlying regular dependency network. These first-order conditional probabilities are typically learned from data by relational learners (section 2.5). RDNs are implemented in a well-developed, public available software package called *Proximity*.

Kersting and DeRaedt [KDR00] introduce Bayesian Logic Programs. This work's motivation is to provide a language which is as syntactically and conceptually simple as possible while preserving the expressive power of works such as Ngo and Haddawy [NH95], Jaeger [Jae97] and PRMs [GFKP01]. According to the authors, this is necessary so one understands the relationship between all these approaches, and also the fundamental aspects of FOPI models.

Fierens at al [FBBR05] define Logical Bayesian Networks (LBNs). LBNs are very similar to Bayesian Logic Programs, with the difference of having both random variables and deterministic logical literals in their language. A logic programming inference process is run for the construction of the Bayesian network, during which logical literals are used, but since they are not random variables, they are not included in the Bayesian network. This addresses the same issue of fixed relationships discussed in the presentation of PRMs, that is, when a set of relationships is deterministically known, we can create random variable nodes in the Bayesian network with significantly fewer ancestors. In the BLPs and LBNs framework, this is exemplified by a rule such as:

$$rating(X) \leftarrow cook(X, Y), training(Y).$$

which has an associated probability, declaring that a restaurant X's rating depends on their cook Y's training. In Bayesian Logic Programs, the instantiations of $cook(X, Y)$ are random variables (just like the instantiations of $rating(X)$ and $training(Y)$). Therefore, since we do not know a priori which Y makes $cook(timpone, Y)$ true, $rating(timpone)$ depends on all instantiations of $cook(timpone, Y)$ and $training(Y)$ and has all of them as parents in the underlying Bayesian network. If in the domain at hand the information of *cook* is deterministic, then this would be wasteful. We could instead determine Y such that $cook(timpone, Y)$, say $Y = joe$, and build the Bayesian network with only the relevant random variable $training(joe)$ as parent of $rating(timpone)$. This is precisely what LBNs do. In LBNs, one would define *cook* as a deterministic literal that would be reasoned about, but not included in the Bayesian network as a random variable. This in fact is even more powerful than the PRMs approach since it deals even with the situation where relationships are not directly given as data, but have to be reasoned about in a deterministic manner.

Santos Costa et al. [CPQC03b] propose an elegant KBMC approach that smoothly leverages an already

existing framework, Constraint Logic Programming (CLP). In regular logic programming, the only constraint over logical variables are equational constraints coming from unification. As explained in section 2.4.2, CLP programs generalize this by allowing other constraints to be stated over those variables. These constraints are managed by special-purpose constraint solvers as the derivation proceeds, and the failure in satisfying a constraint determines the failure of the derivation. The authors leverage CLP by developing a constraint solver on probabilistic constraints expressed as CPTs, and simply plug it into an already existing CLP system. The resulting system can also use available logic programming mechanisms in the CPT specification, making it possible to calculate it dynamically, based on the context, rather than by fixed tables. The probabilistic constraint solver uses a Bayesian network internally in order to solve the posed constraints, so this system is also using an underlying propositional Bayesian network for answering queries. Santos Costa et al. point out [Ang02] as the closest approach to theirs, with the difference that the latter keeps hard constraints on Bayesian variables separate from probabilistic constraints. This allows hard constraints to be solved separately. It is also different in that it does not use conditional independences (like Bayesian networks do), and therefore inference is exponential on the number of random variables.

Markov Logic Networks (MLNs) [RD04] is a recent and rapidly evolving framework for probabilistic logic. Its main distinctions are that it is based on undirected models and has a very simple semantics while keeping the expressive power of first-order logic. The downside to this is that its inference can become quite slow if complex constructs are present.

MLNs consist of a set of weighted first-order formulas and a universe of objects. Its semantics is simply that of a Markov network whose features are the instantiations of all these formulas given the universe of objects. The potential of a feature is defined as the exponential of its weight in case it is true. Fig. 2.2 shows an example.

Formulas can be arbitrary first-order logic formulas, which are converted to clausal form for inference. Converting existentially quantified formulas to clausal form usually involves Skolemization, which requires uninterpreted functions in the language. Since MLNs do not include such functions, existentially quantified formulas are replaced by the disjunction of their groundings (this is possible because the domain is finite). The great expressivity of MLNs allows them to easily represent the same joint distributions represented by other proposed FOPI models. They are also a generalization of first-order logic, to which they reduce when weights are infinite.

Learning algorithms for MLNs have been presented from the beginning. Because learning in undirected models is hard, MLNs use the notion of pseudo-likelihood [Bes75], an approximate but efficient method. When data is incomplete, EM is used.

English / First-Order Logic	Clausal form	Weight
"Smoking causes cancer" $\forall X$ Smokes(X) $\Rightarrow$ Cancer(X)	$\neg$ Smokes(X) $\vee$ Cancer(X)	1.1
"If two people are friends either both smoke or neither does" $\forall X \forall Y$ Fr(X,Y) $\Rightarrow$(Sm(X) $\Leftrightarrow$ Sm(Y))	$\neg$ Friends(X,Y) $\vee$ Smokes(X) $\vee$ Smokes(Y)	1.1

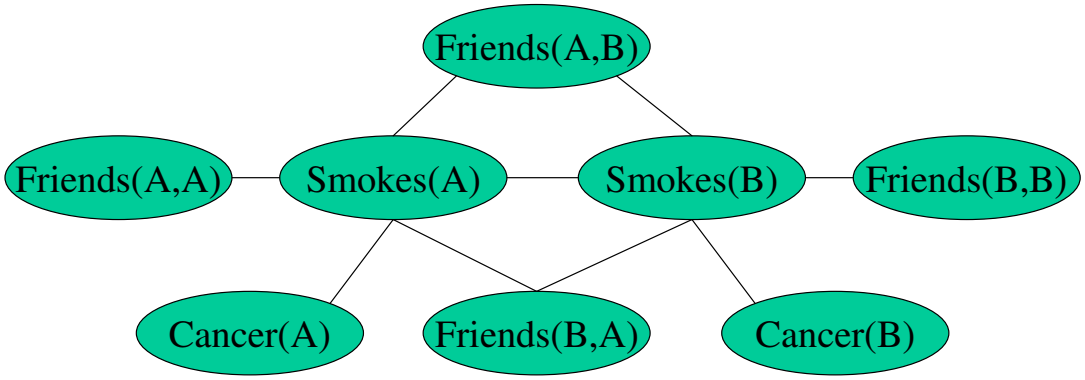

Figure 2.2: A ground Markov network generated from a Markov Logic Network for objects Anna (A) and Bob (B) (example presented in [RD04]).

MLNs are a powerful language and framework accompanied by well-supported software (called *Alchemy*) and which has been applied to real domains. The drawback of its expressivity is potentially very large underlying networks (for example, when existential quantification is used).

Laskey [Las05] presents multi-entity Bayesian networks (MEBNs), a first-order version of Bayesian networks, which rely on generalizing typical Bayesian network representations rather than a logic-like language. A MEBN is a collection of Bayesian network *fragments* involving parameterized random variables. As in the other approaches, the semantics of the model is the Bayesian network resulting from instantiating these fragments. Once they are instantiated, they are put together according to the random variables they share. A MEBN is shown in fig. 2.3.

Laskey's language is indeed quite rich, allowing infinite models, function symbols and distributions on the parameters of random variables themselves. The works focus on defining this language rather than on the actual implementation, which is based on instantiating a Bayesian network containing the parts relevant to the query at hand. It does not provide a detailed account of this process, which can be especially tricky

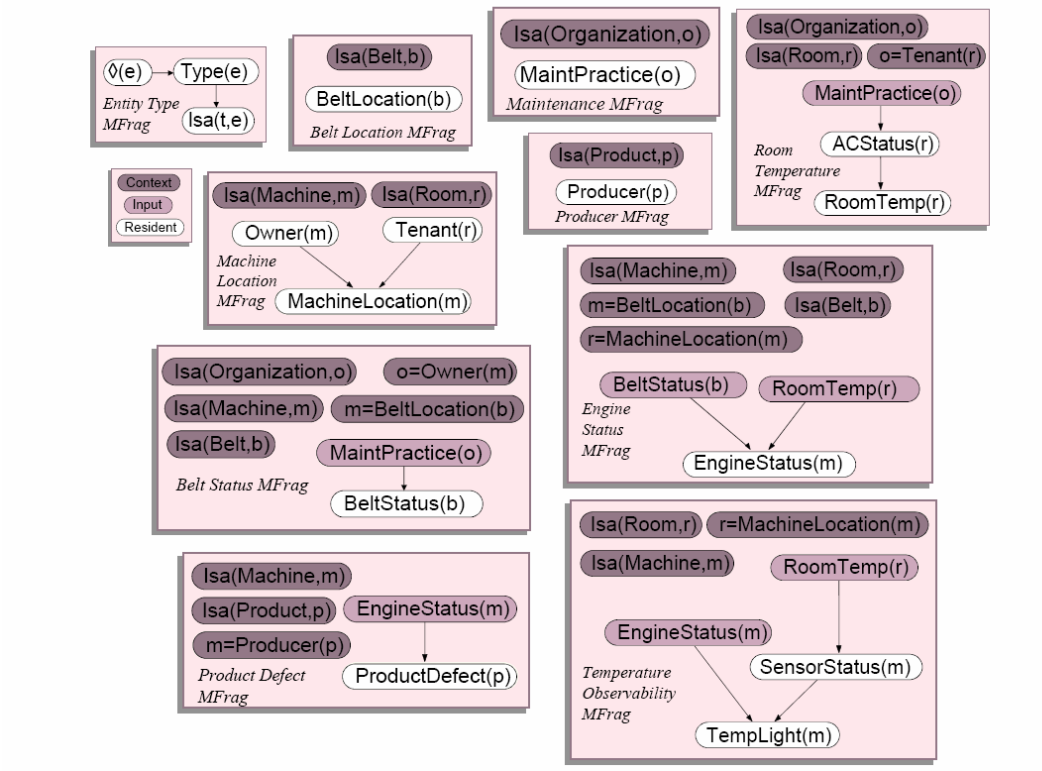

Figure 2.3: An example of a MEBN, as shown in [Las05].

in the case of infinite models.

2.5 Relational Learning

In this section we discuss some first-order models developed from a machine learning perspective.

Machine learning algorithms have traditionally been defined as classification of *attribute-value vectors* [Mit97]. In many applications, it is more natural and convenient to represent data as *graphs*, where each vertex represents an object and each edge represent a relation between objects. Vertices can be labeled with attributes of its corresponding object (unary predicates or binary predicates where the second argument is a simple value; this is similar to PRMs in section 2.4.4), and edges can be labeled (the label can be interpreted as a binary predicate holding between the objects). This gives us the typical data structure representations such as trees, list etc, and collections of objects in general. When learning from graphs, we usually want to form hypotheses that explain one or more of the attributes and (or) relations (the

targets) of objects in terms of its neighbors. Machine learning algorithms which were developed to benefit from this type of representation have often been called *relational*. This is closely associated to probabilistic first-order models, since graph data can be interpreted as a set of ground literals using unary and binary predicates. Because the hypotheses explaining target attributes and relations apply to several objects, it is also convenient to represent the learned hypotheses as quantified (first-order) rules. And because most learners involve probabilities or at least some measure of uncertainty, probabilistic first-order rules provide a natural representation option. Figure 2.4 illustrates these concepts.

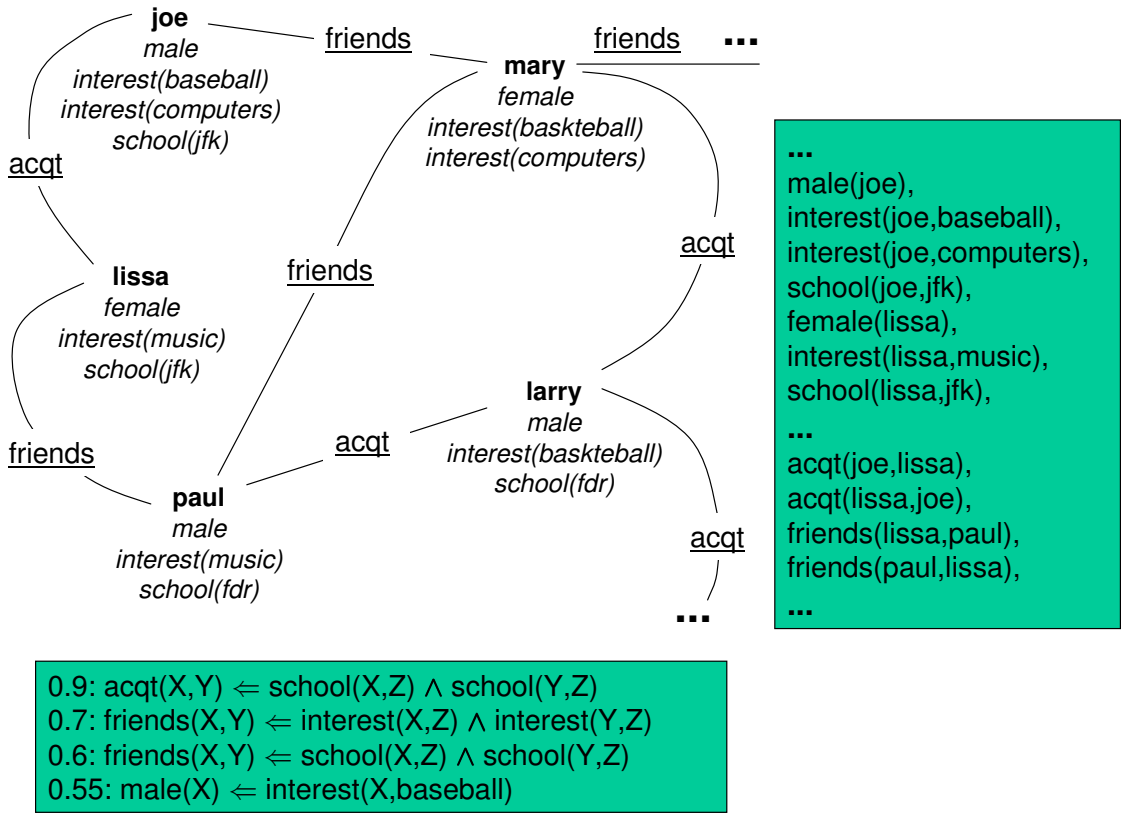

Figure 2.4: A graph structure used as input for relational learning. The same information can be represented as a set of ground literals (right). The hypotheses learned to explain either relations or attributes can be represented as weighted first-order clauses over those literals (below).

We now discuss three forms of relational learning: propositionalization (flattening), Inductive Logic Programming (ILP), and FOPI learning, which can be seen as a synthesis of the two.

2.5.1 Propositionalization

An approach to relational machine learning is that of using a relational structure for generating propositional attribute-value vectors for each of its objects. For this reason, the approach has been called *propositionalization*. Because it transforms graph-like data into vector-like data, it is also often called *flattening*.

Cumby & Roth [CR00] provide a language for transforming relational data into attribute-value vectors. Their concern is not forming a first-order hypothesis, however. They instead keep the attribute-value hypothesis and transform novel data to that representation in order to classify it with propositional learners such as Perceptron. For example, in the case of fig. 2.4, a classifier seeking to learn the relation *acqt* would go through the instances of that predicate and generate suitable attribute-value vectors. The literal $acqt(paul, larry)$ would generate an example with label $acqt(X, Y)$ and features $male(larry)$, $male(paul)$, $interest(larry, basketball)$, $school(larry, fdr)$, $interest(larry, basketball)$, $school(larry, fdr)$ etc, as well as non-ground ones such as $male(X)$, $male(Y)$, $school(X, fdr)$, $school(Y, fdr)$, $school(X, Z)$, $school(Y, Z)$ etc. The literal $acqt(joe, lissa)$ would generate an example with label $acqt(X, Y)$ and features $male(joe)$, $female(lissa)$, $interest(joe, baseball)$, $interest(joe, computers)$, $school(joe, jfk)$,etc, as well as non-ground ones such as $male(X)$, $female(Y)$, $school(X, jfk)$, $school(Y, jfk)$, $school(X, Z)$, $school(Y, Z)$ etc. Note how this reveals abstractions – the examples above share the features $school(X, Z)$ and $school(Y, Z)$, which may be one reason for people being acquaintances in this domain. Should a target depend on specific objects (say, it is much more likely for people at the FDR school to be acquainted to each other) not completely abstracted features such as $school(X, fdr)$ would be preferred by the classifier.

There are many different ways of transforming a graph or set of literals into attribute-value vectors for propositional learners. Each of them will represent different learning biases. Some works in this line are LINUS [LD93], which uses the concept of ij-determinacy (a way of restrict the generalizations of literals) in order to construct hypothesis, 1BC [FL99] and 1BC2 [LF03], which differentiate between predicates representing attributes or relations and construct multiset attributes for example, the set of interests among one's friends), and Relational Bayesian Classifier (RBC) [NJG03], which also uses multiset attribute values with a conditional independence assumption similar to the one used in the Naive Bayes classifier.

One disadvantage of propositionalization is that it performs classification of an object at a time. This prevents the use of possible dependencies between object labels and adds a bias. These dependencies can be used by FOPI algorithms, which perform *joint* inference over all objects at once. Three of FOPI models which have been specifically developed with this in mind are RDNs [Nev06], RMNs [TAK02] and MLNs [RD04].

2.5.2 Inductive Logic Programming

Inductive Logic Programming (ILP) stemmed from the logic programming community with the goal of learning logic programs from data rather than writing them. The choice of logic programming as a hypothesis language restricted the field to deterministic hypotheses; the later incorporation of probabilities to this framework is one of the origins of FOPI models. However, the fact that these algorithms learn from data give them a statistical flavor even in the deterministic case. For example, PROGOL [Mug95a] uses information-theoretic measures for evaluating deterministic hypotheses.

A typical ILP algorithm works by forming hypotheses one clause at a time. For example, if such an algorithm is trying to learn the concept $mother(X, Y)$, it might generate the clause $mother(X, Y) : -female(X)$, since $female(X)$ does add some predictive power to whether X is a mother, and may at a latter step refine it to $mother(X, Y) : -female(X), child(Y, X)$. This is a top-down approach since the most general clause is successively made more specific according to examples. Two examples of work in this line are [Qui90, RD97]. Another approach is bottom-up, best exemplified by Progol [Mug95a], where sets of ground literals are successively generalized in order to increase accuracy.

Some ILP algorithms use propositionalization as a subroutine that learns one rule at a time. LINUS [LD93] transforms its data into attribute-value vectors, applies regular attribute-value learners to it, and then transforms the attribute-value hypothesis into a clause.

Probabilistic ILP (PILP) algorithms (a specific survey can be found at [RK04]) also often grow clauses and then estimate the probabilities (or parameters) associated with those clauses. Some learn an entire logic program at first, and then the parameters, while others learn the parameters as soon as the clauses are learned. In fact, PILP is simply FOPI with learning done with ILP techniques, and many of these works have been mentioned in our FOPI section. For example, MLNs [RD04], PRMs [GFKP01, KP98] and BLPs [KDR01] learn the structure of their models (as opposed to their parameters) through techniques similar to those of ILP. RDNs [Nev06] use relational classifiers as a subroutine to its model learning.

2.6 Restricted First-Order Probabilistic Models

The models presented so far intended to provide a level of expressivity similar to first-order logic. At a minimum, they provide unary and binary predicates arbitrarily applied to a collection of objects. However, there are some probabilistic models showing some first-order aspects, but much more restricted, making them fit for particular tasks.

One such model is Hidden Markov Models (HMMs) [Rab90], in which a sequence of pairs of random

variables represent a *state* and an *observation*. While essentially propositional, HMMs exhibit the sharing of parameters commonly observed in first-order models, since its parameters equally apply to all transitions from one step to the next.

The same sharing of parameters can be observed in related models. Stochastic Context-Free Grammars [LY90] consist of set of production rules were a nonterminal symbol is stochastically replaced by a number of possible sequences of symbols. Again, the rules can be applied at different points and parameters are reused. Dynamic Bayesian Networks [Mur02a] generalize HMMs in that each step is represented by a full Bayesian network rather than just a pair of state and observations.

Other generalizations of these restricted models are Hidden Tree Markov Models [DFG03], where possible states form a tree structure, Logical Hidden Markov Models [KDR00], where each state is represented by a set of logical literals, and Relational Markov Models [ADW02], where each state is also represented by a set of logical literals, but possible arguments follow a taxonomy and induce a lattice on possible literals.

Another simple generalization of propositional models is the plate notation [Bun94], which describes graphical models with parts replicated by a set of indices, indicated by a rectangle involving that part in a diagram. An example is shown in fig. 3.5 (a). The parameters into these parts are then also replicated. Mjolsness [Mjo04] proposes many refinements to this type of notation. The plate notation is commonly used as a tool for descriptions in the literature but does not have a strictly formal characterization.

2.7 Conclusion

First-order probabilistic inference has made much progress in the last twenty years. We believe the main accomplishments have been the clarification the semantics of such languages, as well as a greater understanding on how several different language options relate to each other. In analyzing the works in the area of FOPI, we can distinguish a few main options along which they seem to organize themselves. We now make these aspects more explicit.

2.7.1 Directed vs. Undirected

The decision on using directed or undirected models carries over the the first-order case. While the intelligibility of directed models has favored their use in first-order proposals at first, we can observe a current tendency to use undirected models now [RD04, dSBAR05], or at least directed models without the acyclicity requirement [Nev06]. The reason for this shift is that cycles are even more naturally occurring in first-order domains than in propositional ones. A "natural" conditional probability such as

$P(smoker(X)|friend(X,Y), smoker(Y))$ creates an underlying network with cycles. The use of undirected models seems to be further justified by the fact that they do not rule out directed models. If the given factors encode conditional probabilities that do not involve cycles, they will still represent the correct distribution even if interpreted as an undirected factor (this may however waste the efficiency advantages of directionality).

2.7.2 Deterministic Relations

As mentioned in the presentations of PRMs [GFKP01, KP98] and LBNs [FBBR05], the fact that certain relations are deterministic or given in the data may significantly decrease the size of CPTs (or factor table) involved in a model. This is an important practical issue that needs to be kept in mind when implementing FOPI systems.

2.7.3 The Trade-off between Language and Algorithm

Some FOPI proposals focus on rich languages that allow the user to indicate domain restrictions which can be exploited for efficiency. Example of such systems are Ngo and Haddawy's Probabilistic Logic Programming [NH95], PRMs [GFKP01, KP98], LBNs [FBBR05]. Other solutions propose extremely simple yet expressive languages, relying on inference algorithms to exploit domain structure (sometimes guided by extra-language directives). The most typical examples are BLPs [KDR00] and MLNs [RD04].

This choice reflects a trade-off between language and inference algorithm complexities. Complex languages bring built-in optimization hints; for example, PRMs have attributes (with a value) and relations, even though both could be regarded as binary predicates. By indicating that a binary predicate is an attribute, the user implicitly indicates the most efficient ways of using it, which are distinct from the way a relation is used. To obtain the same effect, an algorithm using only a single general notion of binary predicates would have to either find out or be told about how to use each of them in the most efficient manner.

Our particular view is that FOPI languages will tend to become simpler, leaving the complexities to be dealt with by software. This reflects the evolution of programming languages, that have increasingly left efficiency details to be dealt with by compilers and directives rather than by the language itself, leaving the latter at a higher level.

2.7.4 Function Symbols

Functions symbols pose complicated problems for FOPI models. If used recursively, they can create the need to consider infinite models, preventing straightforward solutions such as creating an underlying graphical model, or computing all possible worlds.

Since it is easier to deal with infinite models by approximation, it is perhaps not surprising that sampling solutions such as PRIM [SK97] and BLOG [MMR+05] can use function symbols. Exact methods, or guaranteed-bound approximation methods, are yet to make full use of function symbols.

Other topics that have not been much explored in the literature are the need for open worlds (only BLOG [MMR+05], to our knowledge, uses it) and *guided* underlying graphical model construction, that is, having the underlying graphical model be constructed in a way that maximizes the accuracy of an anytime computation. This would delay the expensive construction of sections of the network that do contribute to the answer (thus not being eliminable due to strict irrelevance) while mattering little.

Chapter 3

Lifted First-Order Probabilistic Inference

This chapter presents our main contribution, the FOVE algorithm for lifted FOPI inference. The FOVE algorithm receives a first-order probabilistic model, in a language defined in section 3.2, and a query (a set of "ground" random variables), and computes the marginal probability of the query given the model.

What distinguishes the FOVE algorithm from other FOPI algorithms is that it often performs inference directly at the first-order level, without generating a propositional model first. This allows the algorithm to eliminate entire classes of random variables without considering them individually, greatly increasing efficiency in large domains where propositionalization would create a much larger model.

3.1 Propositional Markov Networks and Variable Elimination

3.1.1 Markov Networks

A Markov network is a graphical model defined on random variables $\mathbf{V} = (V_1, \ldots, V_n)$. It consists of a collection of *potential functions* Φ, with each $\phi \in \Phi$ defined on a tuple $\mathbf{A}_\phi$ of random variables in $\mathbf{V}$. These potential functions can also be referred to as *factors*.

A Markov network can be represented as a graph with random variables as vertices and edges linking random variables which are inputs to a same potential function, as illustrated in figure 3.1(a). In this representation, potential functions apply to every maximal clique in the graph. Alternatively, we can explicitly represent potential functions as a different type of vertex and link each random variable to the potential functions for which it is an input, as shown in 3.1(b). This type of representation is called a *factor network* and is the one we use for the remainder of the thesis.

A potential function maps each value assignment of its random variables to a non-negative real number representing the degree of compatibility between the assigned values. This number is the *potential* of that assignment given by that particular potential function.

Although a potential does not have a probabilistic meaning per se, a Markov network defines a joint

distribution on its random variables. The probability of an assignment $\mathbf{v}$ to $\mathbf{V}$ is defined as

$$P(\mathbf{v}) = 1/Z \prod_{\phi \in \Phi} \phi(\mathbf{a}_\phi)$$
$$Z = \sum_{\mathbf{v}' \in Val(\mathbf{V})} \prod_{\phi \in \Phi} \phi(\mathbf{a}_\phi)$$

where $\mathbf{a}_\phi$ is the assignment to $\mathbf{A}_\phi$ according to the current global assignment, and $Val(\mathbf{V})$ is the set of possible assignments to $\mathbf{V}$.

We can express the above more succinctly by indicating that the probability $P(\mathbf{v})$ is proportional to the product of potentials:

$$P(\mathbf{v}) \propto \prod_{\phi \in \Phi} \phi(\mathbf{a}_\phi).$$

3.1.2 Variable Elimination

Given a Markov network and its induced joint probability, a problem of interest is that of *Belief Assessment*, which consists of computing the marginal probability of a subset $\mathbf{Q}$ of its random variables, defined as

$$P(\mathbf{q}) \propto \sum_{\mathbf{v} \setminus \mathbf{q}} \prod_{\phi \in \Phi} \phi(\mathbf{a}_\phi)$$

where $\mathbf{v} \setminus \mathbf{q}$ ranges over all assignments in $Val(\mathbf{V} \setminus \mathbf{Q})$[1]. This sum is intractable in general because the size of $Val(\mathbf{V} \setminus \mathbf{Q})$ is exponential in the size of $\mathbf{V} \setminus \mathbf{Q}$. A more efficient algorithm, called Variable Elimination (VE) [ZP94] or Bucket Elimination [Dec99], takes advantage of the model's structure and factors potential functions out of certain summations. Given an ordering, $\{V_{o_1}, \ldots, V_{o_m}\}$, of $\mathbf{V} \setminus \mathbf{Q}$, we can write

$$P(\mathbf{q}) \propto \sum_{\mathbf{v}_{o_1}} \cdots \sum_{\mathbf{v}_{o_m}} \prod_{\phi \in \Phi} \phi(\mathbf{a}_\phi)$$
$$= \prod_{\phi \in \Phi_0} \phi(\mathbf{a}_\phi) \sum_{\mathbf{v}_{o_1}} \prod_{\phi \in \Phi_1} \phi(\mathbf{a}_\phi) \cdots \sum_{\mathbf{v}_{o_m}} \prod_{\phi \in \Phi_m} \phi(\mathbf{a}_\phi)$$

[1]Posterior probabilities given a set of observations can be computed by adding potential functions on observed random variables that assign non-zero potentials to the observed values only

where $\Phi_j \subseteq \Phi$, for $j > 0$, is the set of factors with V_{o_j} but none of $V_{o_k}, k > j$ in their inputs, and $\Phi_0 \subseteq \Phi$ is the set of factors with no V_{o_j} as input. Then, we calculate the innermost summation and form a new factor ϕ':

$$\phi'(\mathbf{a}_{\phi'}) = \sum_{\mathbf{v}_{o_m}} \prod_{\phi \in \Phi_m} \phi(\mathbf{a}_\phi)$$

where $\mathbf{A}_{\phi'}$ is $\left(\bigcup_{\phi \in \Phi_{o_m}} \mathbf{A}_\phi\right) \setminus \mathbf{V}_{o_m}$. In the next step we add ϕ' to the appropriate Φ_j and the algorithm resumes. It stops when only Φ_0 remains, the product of which provides a marginal distribution on $\mathbf{Q}$.

Graphically, VE corresponds to eliminating a random variable node from the network and collapsing its neighboring potential functions. Figure 3.1(c) shows the resulting factor network after the elimination of random variable *sick* from the network in figure 3.1(b). This continues until all random variables not in the query are eliminated. We are then left with a network on the query alone that represents its marginal directly.

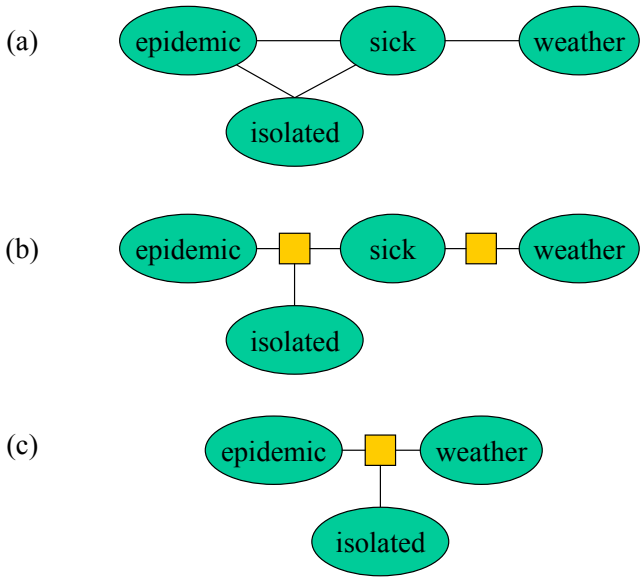

Figure 3.1: (a) a graphical representation of a Markov network, (b) its alternative representation known as a factor network, where squares represent factors (potential functions) on the random variables (circles) to which they connect, and (c) the resulting factor network after the Variable Elimination of random variable *sick*.

3.2 FOPI Language, Semantics and Inference Problem

Like Markov Networks, First-Order Probabilistic Models (FOPMs) are essentially defined by a set of factors. However, unlike them, these factors are defined over *parameterized* random variables, and for this reason we call them **parfactors** (following [Poo03]).

Given a universe of objects over which the parameters range, we can generate regular propositional factors from a parfactor by replacing its parameters by particular objects. A parfactor is therefore a compact representation of a set of regular factors, and a FOPM is a compact representation of a Markov network composed by all instantiations of all of its parfactors. Figure 3.2 shows an example of a FOPM and its meaning.

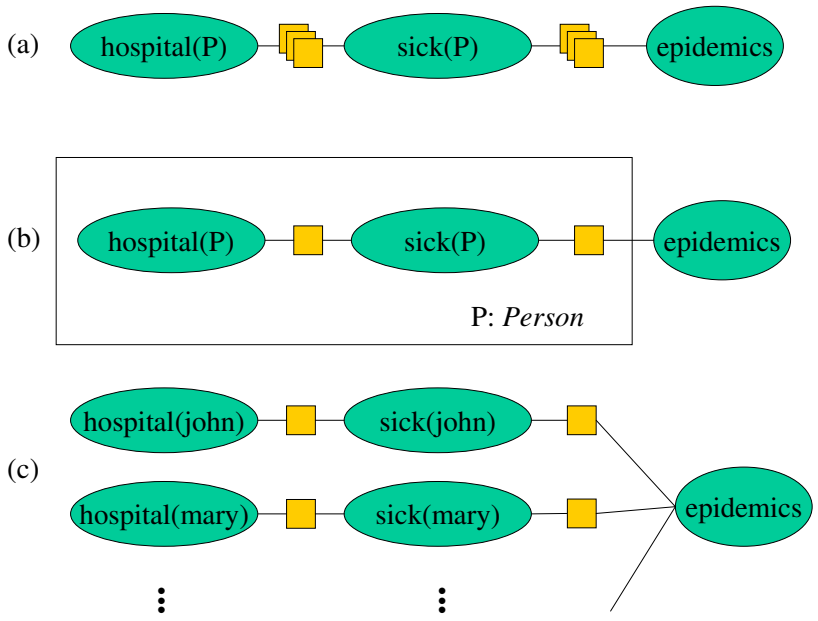

Figure 3.2: (a) a graphical representation of a first-order probabilistic model (where piled squares represent parfactors), (b) an equivalent plate representation and (c) its grounding, where squares represent factors.

Due to the correspondence to logic concepts, we call a parameterized random variable an **atom**, and a parameter a **logical variable** (as opposed to *random* variables). We also call the functors of atoms **predicates**. Even though we informally refer to atoms as "parameterized random variables", they are not, technically speaking, random variables, but stand for classes of them. A ground atom, however, denotes a random variable. Sometimes we call random variables **ground** to emphasize their correspondence to ground atoms.

Logical variables are typed, with each type being a finite set of objects. We denote the **domain**, or **type**, of a logical variable X by D_X and its cardinality by $|X|$. In our examples, unless noted, all logical variables

have the same type. Each predicate p also has its domain, D_p, which is the set of values that each of the random variables with that predicate can take.

Formally, a **parfactor** g is a tuple (ϕ_g, A_g, C_g), where ϕ_g is a potential function defined over atoms A_g to be instantiated by all substitutions of its logical variables satisfying a constraint C_g. A **constraint** is a pair (F, V) where F is an equational formula on logical variables and V is the set of logical variables to be instantiated (some of them may not be in the formula). We sometimes denote a constraint by its formula F alone, when the set of logical variables V is clear from context. Tautological formulas are represented by $\top$. For example, the parfactor $(\phi, (p(X), q(X, Y)), (X \neq a, \{X, Y\}))$ applies ϕ to all instantiations of $(p(X), q(X, Y))$ by substitutions of X and Y satisfying $X \neq a$. We denote the set of substitutions satisfying C by $[C]$.

While we are neutral as to how the potential functions are actually specified, logical formulas seem to be a convenient choice. For example, a weighted formula $0.7 : epidemic(D) \Rightarrow sick(P, D)$ might represent a potential function $\phi(epidemic(D), sick(P, D))$ with potential 0.7 for assignments in which the formula is true. This allows us to specify FOPMs by sets of weighted logical formulas that are intuitive and simple to read, and is the approach taken by Markov Logic Networks ([RD04]).

The **projection** $C_{|L}$ of a constraint $C = (F, V)$ **onto a set of logical variables** L is a constraint equivalent to $(\exists L' F, L)$ for $L' = V \setminus L$. Intuitively, $C_{|L}$ describes the conditions posed by C on L alone, that is, the possible substitutions on L that are part of substitutions in $[C]$. For example, $(X \neq a \wedge X \neq Y \wedge Y \neq b, \{X, Y\})_{|\{X\}} = (X \neq a, \{X\})$. FOVE uses a constraint solver which is able to solve several constraint problems, such as determining the number of solutions of a constraint and its projection onto sets of logical variables.

In certain contexts we wish to describe the class of random variables instantiated from an atom with constraints on its logical variables (for example, the set of random variables instantiated from $p(X, Y)$, with $X \neq a$). We call such pairs of atoms and constraints **constrained atoms**, or **c-atoms**. The **c-atoms of a parfactor** is the set of c-atoms formed by its atoms and its constraint.

Let α be a parfactor, c-atom, constraint or a set of those. We define $RV(\alpha)$ to be the set of (ground) random variables specified by α, and $\alpha\theta$ denotes the result of applying a substitution θ to α. $[C_g]$ is also denoted by Θ_g.

A FOPM is specified by a set of parfactors G and the types of its logical variables. Its semantics is a joint distribution defined on $RV(G)$ by the Markov Network formed by all the instantiations of parfactors.

Thus it is proportional to the product of all instantiated parfactors:

$$P(RV(G)) \propto \prod_{g \in G} \prod_{\theta \in \Theta_g} g\theta$$

For convenience, we denote $\prod_{\theta \in \Theta_g} g\theta$ by $\Phi(g)$, and $\prod_{g \in G} \Phi(g)$ by $\Phi(G)$. Therefore we can write the above as $P(RV(G)) \propto \Phi(G)$.

The most important inference task in graphical models is marginalization. For FOPMs, it takes the following form: given a set of ground random variables Q, calculate

$$P(Q) \propto \sum_{RV(G) \setminus Q} \Phi(G) \tag{3.1}$$

where the summation ranges over all assignments to $RV(G) \setminus Q$. Posterior probabilities can be calculated by representing evidence as additional parfactors on the evidence atoms.

The FOVE algorithm makes the simplifying assumption that the set of instantiations of any pair of c-atoms in either the model or the query are either identical or disjoint. In other words, we can have c-atoms $(p(X), X \neq a)$,$(p(Y), Y \neq a)$ and $p(a)$ in a model, but not $p(Y)$ and $p(a)$, for example, because $RV(p(a)) \subset RV(p(Y))$ but $RV(p(a)) \neq RV(p(Y))$. When a model and query do not satisfy this condition, they need to be **shattered**. Shattering is explained in section 3.6.2.

3.3 The First-Order Variable Elimination (FOVE) Algorithm

Computing (3.1) directly is intractable since it would take exponential time in the number of random variables in $RV(G) \setminus Q$. This is the case even for the propositional case, which is the reason why algorithms have been developed that take advantage of independences represented in the model in order to compute marginals more efficiently. One of these algorithms is Variable Elimination (VE) ([ZP94]). First-Order Variable Elimination (FOVE) is a first-order generalization of Variable Elimination. While VE eliminates a random variable at a time, FOVE eliminates a c-atom, or set of c-atoms, at each step. By eliminating a c-atom, we implicitly eliminate all of its instantiations at the same time. Let E be a set of c-atoms to be eliminated from a FOPM with a set G of parfactors. Let $G_E, G_{\neg E} \subseteq G$ be the sets of parfactors depending and not depending on E, respectively. Then

$$\sum_{RV(G) \setminus Q} \Phi(G) = \sum_{(RV(G) \setminus RV(E)) \setminus Q} \Phi(G_{\neg E}) \sum_{RV(E)} \Phi(G_E)$$

We later show operations computing a parfactor g' such that $\sum_{RV(E)} \Phi(G_E) = \Phi(g')$. Once we have g', the right-hand side of the above is equal to

$$\sum_{(RV(G)\setminus RV(E))\setminus Q} \Phi(G_{\neg E})\Phi(g') = \sum_{(RV(G)\setminus RV(E))\setminus Q} \Phi(G_{\neg E} \cup \{g'\}) = \sum_{RV(G')\setminus Q} \Phi(G')$$

where $G' = G_{\neg E} \cup \{g'\}$. In other words, we have reduced the original marginalization to a smaller instance that does not include $RV(E)$. This is repeated until only Q is left.

A crucial difference between VE and FOVE is elimination ordering. VE eliminates random variables according to an ordering given a priori. In FOVE, eliminating certain c-atoms may require eliminating some other c-atoms first, so it may be the case that some c-atoms are not eliminable at all times (these conditions will be clarified later). Because parfactors and c-atoms are sometimes changed and reorganized during the algorithm, it is not a simple matter to choose an ordering in advance. Instead, the elimination ordering is dynamically determined.

Before we move on explaining the operations for calculating $\sum_{RV(E)} \Phi(G_E) = \Phi(g')$, we mention that, in fact, they only calculate $\sum_{RV(E)} \Phi(g)$ for a single parfactor g. This is not a problem because the operation of **fusion**, covered in section 3.6.1, calculates g such that $\Phi(g) = \Phi(G)$ for any set of parfactors G.

3.3.1 Counting Elimination

We first show counting elimination on a specific example and later generalize it. Consider the summation

$$\sum_{RV(p(X))} \prod_{X,Y} \phi(p(X), p(Y))$$

where p is a boolean predicate. (Note that the X used under the summation is not the same X used by the product. $RV(p(X))$ is shorthand for all assignments over the set $\{p(X) : X \in D_X\}$, so X is locally used. In fact, we could have written $RV(p(Y))$, or even $RV(p(Z))$, to the same effect. We choose to use X or Y to make the link with the atom in the parfactor more obvious.)

Counting elimination is based on the following insight: because a parfactor will typically only evaluate to a few different potentials, large groups of its instantiations will evaluate to the same potential. So the summation is rewritten

$$\sum_{RV(p(X))} \phi(0,0)^{|(0,0)|}\phi(0,1)^{|(0,1)|}\phi(1,0)^{|(1,0)|}\phi(1,1)^{|(1,1)|}$$

where $|(v_1, v_2)|$ indicates the number of possible choices for X and Y so that $p(X) = v_1$ and $p(Y) = v_2$ given the current assignment to $RV(p(X))$. These partition sizes can be calculated by a combinatorial, or counting, argument. Assume we know $\vec{N}_p$, a vector of integers that indicates how many random variables in $RV(p(X))$ are currently assigned a particular value, that is, $\vec{N}_{p,i} = |\{r \in RV(p(X)) : r = i\}|$ for each $i \in D_p$. Naturally, $\sum_i \vec{N}_{p,i} = |RV(p(X))|$. Then there are $\vec{N}_{p,v_1}$ possible values for X (so that $p(X) = v_1$) and $\vec{N}_{p,v_2}$ distinct possible values for Y (so that $p(Y) = v_2$), so $|(v_1, v_2)| = \vec{N}_{p,v_1}\vec{N}_{p,v_2}$.

We take advantage of the fact that the values $|(v_1, v_2)|$ do not depend on the particular assignments to $RV(p(X))$, but only on $\vec{N}_p$. This allows us to iterate over the *groups* of assignments with the same $\vec{N}_p$ and do the calculation for the entire group. We also take into account the group size, which is provided by the binomial coefficient of $\vec{N}_p$, $\binom{|RV(p(X))|}{\vec{N}_{p,0}}$ (or, equivalently, $\binom{|RV(p(X))|}{\vec{N}_{p,1}}$). We then have

$$\sum_{\vec{N}_p} \binom{|RV(p(X))|}{\vec{N}_{p,0}} \prod_{(v_1,v_2)} \phi(v_1, v_2)^{\vec{N}_{p,v_1}\vec{N}_{p,v_2}}$$

which has a number of terms linear in $|RV(p(X))|$, as opposed to the previous exponential number.

Figure 3.3 shows the input and output, in plate notation, of the counting elimination of $RV(sick(P))$ from a parfactor on $epidemic, sick(P_1), sick(P_2)$.

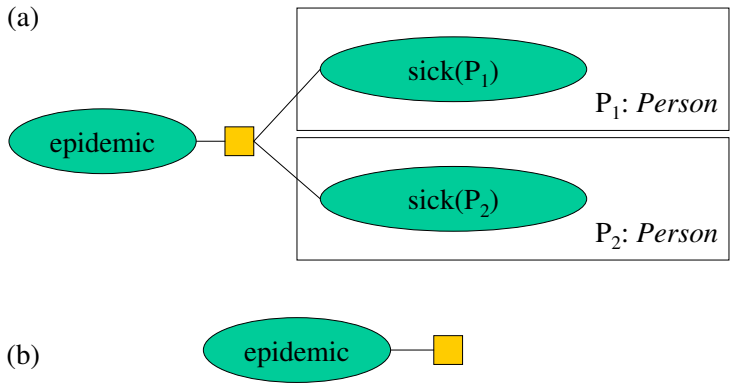

Figure 3.3: Plate representation of counting elimination (a) before and (b) after. Note that the grounding of $sick(P_1)$ and $sick(P_2)$ is the same, which is not standard in plate representation.

Counting elimination is not a universal method. The counting argument presented above requires that

there be little interaction between the logical variables of atoms. If a parfactor is on $p(X,Y), q(X,Z)$, for example, the counting argument does not work because the choices for (X,Z) depend on the particular X chosen for $p(X)$; we can no longer compute number of choices using counters alone but need to know the particular assignment to $RV(p(X))$. Generally, under counting elimination choices for one atom cannot constrain the choices for another atom (there are exceptions to this rule, as for example *just-different* atoms, presented in ([dSBAR06])).

We now give the formal account of counting elimination, starting by some preliminary definitions.

First, we define the notion of **independent atoms given a constraint**. Intuitively, this happens when choosing a substitution for the logical variables of one atom does not change the possible choices of substitutions for the other atom. Let $\bar{X}_1$ and $\bar{X}_2$ be two sets of logical variables such that $\bar{X}_1 \cup \bar{X}_2 \subseteq V$. $\bar{X}_1$ is *independent from* $\bar{X}_2$ *given* C if, for any substitution $\theta_2 \in [C_{|\bar{X}_2}]$, $C_{|\bar{X}_1} \Leftrightarrow (C\theta_2)_{|\bar{X}_1}$. $\bar{X}_1$ and $\bar{X}_2$ are *independent given* C if $\bar{X}_1$ is independent from $\bar{X}_2$ given C and vice-versa. Two *atoms* $p_1(\bar{X}_1)$ and $p_2(\bar{X}_2)$ are *independent given* C if $\bar{X}_1$ and $\bar{X}_2$ are independent given C.

Finally, we define **multinomial counters**. Let a be a c-atom with domain D_a. Then the *multinomial counter of* a $\vec{N}_a$ is a vector where $\vec{N}_{a,j}$ indicates how many instantiations of a are assigned the j-th value in D_a. The *multinomial coefficient* $\vec{N}_a! = \frac{(\vec{N}_{a,1}+\cdots+\vec{N}_{a,|D_a|})!}{\vec{N}_{a,1}!\ldots\vec{N}_{a,|D_a|}!}$ is a generalization of binomial coefficients and indicates how many assignments to $RV(a)$ exhibit the particular value distribution counted by $\vec{N}_a$.

Counters can be applied to *sets* of c-atoms with the same general meaning. The set of multinomial counters for a set of c-atoms A is denoted $\vec{N}_A$, and the product $\prod_{a\in A} \vec{N}_a!$ of their multinomial coefficients is denoted $\vec{N}_A!$.

Theorem 1 (Counting Elimination)**.** Let g be a shattered parfactor and $E = \{E_1, \ldots, E_k\}$ be a subset of A_g such that $RV(E)$ is disjoint from $RV(A_g \setminus E)$, $A' = A_g \setminus E$ are all ground, and where each pair of atoms is independent given C_g. Then

$$\sum_{RV(E)} \prod_{\theta\in\Theta_g} \phi(A_g\theta) = \sum_{\vec{N}_E} \vec{N}_E! \prod_{v\in D_E} \phi(v, A')^{\#(v,\vec{N}_E)}$$

$$\text{where } \#(v, \vec{N}_E) = \prod_{i=1}^{k} \vec{N}_{E_i,v_i}$$

Proof.

$$\begin{aligned}\sum_{RV(E)} \Phi(g) &= \sum_{RV(E)} \prod_{\theta \in \Theta_g} \phi(A_g\theta) \\ &= \sum_{RV(E)} \prod_{v \in D_E} \phi(v, A')^{|\{\theta \in \Theta_g : A_g\theta = v\}|} \\ &= \sum_{RV(E)} \prod_{v \in D_E} \phi(v, A')^{\#(v, \vec{N}_E)} \\ &= \sum_{\vec{N}_E} \vec{N}_E! \prod_{v \in D_E} \phi(v, A')^{\#(v, \vec{N}_E)}\end{aligned}$$

□

Counting elimination brings a significant computational advantage because iterating over assignments is exponential in $|RV(E)|$ while doing so over groups of assignments is only polynomial in it.

It is important to notice that E must contain all non ground c-atoms in g. Also, if all c-atoms in g are ground, E can be any subset of them and we will have a simple propositional summation, the same used in VE (counters over 1-random variable c-atoms reduce to ordinary assignments).

3.3.2 Inversion

Counting elimination requires a parfactor's atoms to be independent given its constraints. In particular, logical variables shared between atoms may render them dependent on each other. In some of these cases, the operation of *Inversion* can be applied. In fact, even in cases in which counting elimination can be applied, it is advantageous to apply Inversion first, if possible, for efficiency reasons.

Let us consider a couple of examples before we formalize Inversion. Consider the following, where D assumes a value from a set of diseases $\{d_1, \ldots, d_n\}$ and P assumes a value from a set of people $\{p_1, \ldots, p_m\}$:

$$\sum_{RV(sick(D,P))} \prod_{D,P} \phi_1(epidemics(D), sick(D,P))\phi_2(sick(D,P), hospital(P))$$

(by abbreviating the predicate names and using fusion (section 3.6.1) to create a single parfactor)

$$\begin{aligned}&= \sum_{RV(s(D,P))} \prod_{D,P} \phi(e(D), s(D,P), h(P)) \\ &= \sum_{s(d_1,p_1)} \sum_{s(d_1,p_2)} \cdots \sum_{s(d_n,p_m)} \phi(e(d_1), s(d_1,p_1), h(p_1))\phi(e(d_1), s(d_1,p_2), h(p_2)) \ldots \phi(e(d_n), s(d_n,p_m), h(p_m))\end{aligned}$$

(by observing that $\phi(e(d_i), s(d_i, p_j), h(p_j))$ depends on $s(d_i, p_j)$ only)

$$= \sum_{s(d_1,p_1)} \phi(e(d_1), s(d_1, p_1), h(p_1)) \cdots \sum_{s(d_n,p_m)} \phi(e(d_n), s(d_n, p_m), h(p_m))$$

(by observing that only $\phi(e(d_i), s(d_i, p_j), h(p_j))$ depends on $s(d_i, p_j)$)

$$= \Big(\sum_{s(d_1,p_1)} \phi(e(d_1), s(d_1, p_1), h(p_1)) \Big) \ldots \Big(\sum_{s(d_n,p_m)} \phi(e(d_n), s(d_n, p_m), h(p_m)) \Big)$$
$$= \prod_{D,P} \sum_{s(D,P)} \phi(e(D), s(D, P), h(P))$$

(by observing that only the summation is the same for all D, P)

$$= \prod_{D,P} \phi'(e(D)).$$

Inversion works by establishing a one-to-one correspondence between parfactor instantiations and summations. If the summation were on the instantiations of $e(D)$, such correspondence would not be possible because there would be less summations (n of them) than parfactor instantiations ($n * m$ of them).

Figure 3.4 illustrates why inversion elimination is faster than regular VE. If we have two parfactors, one on *epidemic* and $sick(P)$ and another on $sick(P)$ and *some_hospital*, regular VE will eliminate each of the (possibly many) instances of $sick(P)$. Inversion elimination, on the other hand, will apply a single elimination step on the $sick(P)$ instead.

It is illuminating to note that, in the context of plate graphical models [Bun94], inversion elimination is the same as eliminating a random variable inside an innermost plate. This is exemplified in figure 3.5.

Even if we have the same number of summations and parfactor instantiations, we may not have the one-to-one correspondence that allows factoring out. For example, we cannot use Inversion on $p(X, Y)$ for a parfactor on $p(X, Y), p(Y, X)$ because for any pair of objects o_i, o_j, neither of the parfactor instantiations $p(o_i, o_j)$ and $p(o_j, o_i)$ can be factored out of the innermost of $\sum_{p(o_i,o_j)}$ and $\sum_{p(o_j,o_i)}$.

In the Inversion example above, the resulting inner summation was propositional. Inversions resulting in propositional summations were called *Inversion Elimination* in ([dSBAR05, dSBAR06]). In the next example, the inner summation is one computed by counting elimination.

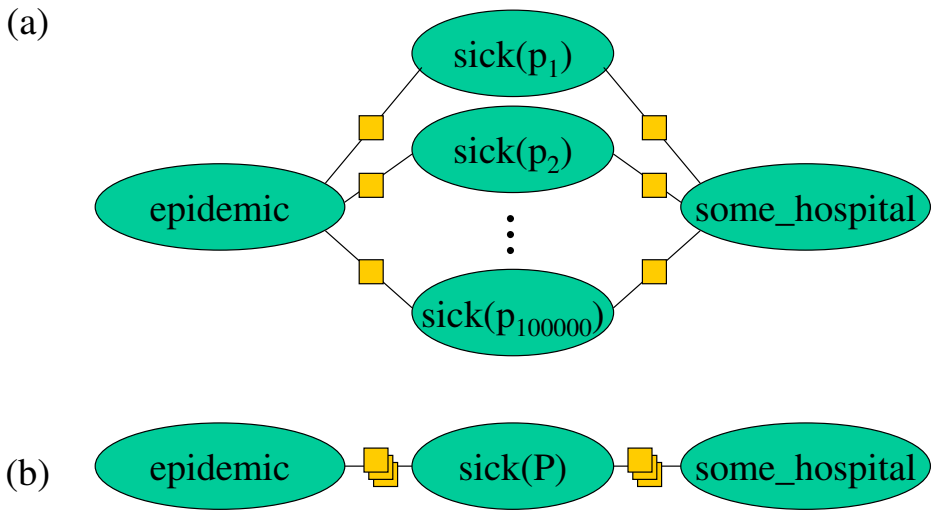

Figure 3.4: Regular VE needs to eliminate each of the instances of $sick(P)$ (a), while Inversion Elimination eliminates $sick(P)$ in a single step.

Suppose we want to calculate

$$
\begin{aligned}
&\sum_{RV(p(X,Y))} \prod_{X,Y,Z} \phi(p(X,Y), p(X,Z)) \\
&= \sum_{RV(p(X,Y))} \prod_{X} \prod_{Y,Z} \phi(p(X,Y), p(X,Z)) \\
&= \sum_{RV(p(o_1,Y))} \cdots \sum_{RV(p(o_n,Y))} \prod_{Y,Z} \phi(p(o_1,Y), p(o_1,Z)) \cdots \prod_{Y,Z} \phi(p(o_n,Y), p(o_n,Z)) \\
&= \Big(\sum_{RV(p(o_1,Y))} \prod_{Y,Z} \phi(p(o_1,Y), p(o_1,Z)) \Big) \ldots \Big(\sum_{RV(p(o_n,Y))} \prod_{Y,Z} \phi(p(o_n,Y), p(o_n,Z)) \Big) \\
&= \prod_{X} \sum_{RV(p(X,Y))} \prod_{Y,Z} \phi(p(X,Y), p(X,Z))
\end{aligned}
$$

Because X is now bound before the summation, it works as a constant (whose exact identity is irrelevant), and so it is not included in the counting argument. The counting argument now involves only Y and Z and is actually very similar to our original counting argument example. For that reason, the above is equal to $\prod_X \phi'()$, for ϕ' the result of counting elimination.

Inversions resulting in counting elimination problems only invert on a subset of the parfactor's logical

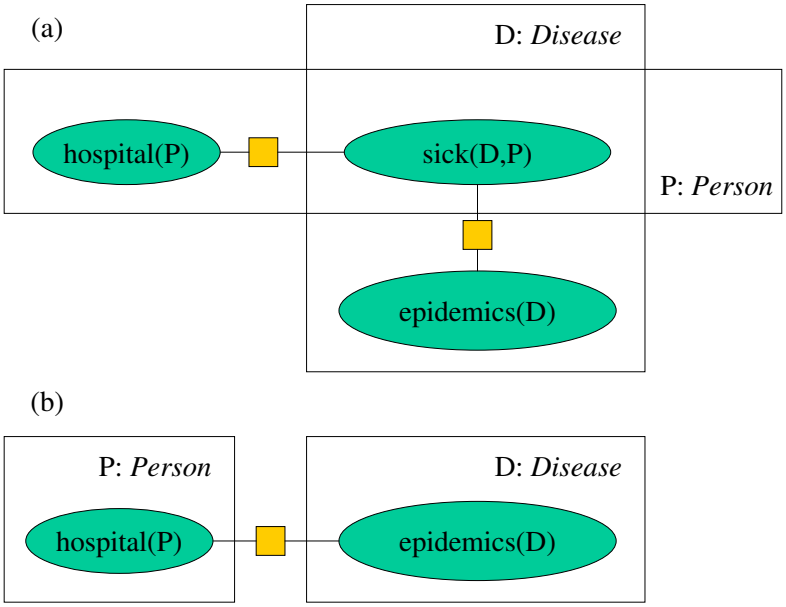

Figure 3.5: Inversion elimination corresponds to the elimination of a random variable schema in an innermost plate in a plate graphical model, here represented by (a) before and (b) after the elimination. Models with crossing plates are exemplified in [Las05].

variables. For this reason, they have been called *Partial Inversions* in our previous work. Note however that, since propositional sums are a trivial case of counting elimination, both Inversion operations can be unified into one. This is what we do in the formalization below.

Uniform Solution Counting Partition (USCP)

Before we present the theorem formalizing Inversion, we touch a last issue. Consider the inversion of X resulting in the expression

$$\prod_{X} \sum_{RV(p(X,Y))} \prod_{Y \neq X, Z \neq X, Y \neq a, Z \neq a} \phi(p(X,Y), p(X,Z)).$$

The summation can be done by counting elimination since X is bound. However, it will depend on $|RV(p(X,Y))|$, but that depends on whether $X = a$ or not. One needs to split the expression according to cases $X = a$ and $X \neq a$:

$$\Big(\sum_{RV(p(a,Y))} \prod_{Y \neq a, Z \neq a} \phi(p(a,Y), p(a,Z)) \Big) \times \Big(\prod_{X \neq a} \sum_{RV(p(X,Y))} \prod_{Y \neq X, Z \neq X, Y \neq a, Z \neq a} \phi(p(X,Y), p(X,Z)) \Big)$$

and then proceed as usual.

In general, one needs to consider the **uniform solution counting partition (USCP)**. of the inverted logical variables with respect to an original constraint system. The USCP $U_L(C)$ of a set of logical variables L with respect to a constraint C is a set of constraints $\{C_1, \ldots, C_k\}$ such that $\{[C_i]\}_i$ forms a partition of $[C_{|L}]$ and

$$\forall i \cdot \forall \theta', \theta'' \in [C_i] \cdot |[C\theta']| = |[C\theta'']|,$$

that is, the number of solutions for the constraint conditioned on L is the same for each of the components C_i.

A more complete treatment of USCPs is given in appendix A.

Inversion Formalization

Theorem 2 (Inversion)**.** Let g be a shattered parfactor, L a set of logical variables and E a set of c-atoms such that $RV(E)$ and $RV(A_g \setminus E)$ are disjoint. If

$$\forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(e_k\theta_i) \cap RV(e_l\theta_j) = \emptyset. \tag{3.2}$$

then

$$\sum_{RV(E)} \Phi(g) = \prod_{C \in U_L(C_g)} \Phi(g_C)$$

where g_C is the parfactor $(\phi_{g'}, A_{g'}, C \wedge C_{g'})$, using g' defined by the recursive computation $g'\theta = \sum_{RV(E\theta)} \Phi(g\theta)$, for θ an arbitrary element of $[C]$ (by the definition of USCP, it does not matter which).

Proof. Let $E = \{e_1, \ldots, e_n\}$, $[C_{g|L}] = \{\theta_1, \ldots, \theta_m\}$. Below, we decompose C_g into the part w.r.t. L and the

remaining logical variables:

$$
\begin{aligned}
&\sum_{RV(E)} \Phi(g) = \sum_{RV(E)} \prod_{\theta \in \Theta_g} \phi(A_g\theta) \\
&= \sum_{RV(E)} \prod_{\theta \in [C_{g|L}]} \prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \\
&= \sum_{RV(e_1)} \cdots \sum_{RV(e_n)} \prod_{\theta \in [C_{g|L}]} \prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \\
&= \sum_{RV(e_1\theta_1)} \cdots \sum_{RV(e_n\theta_1)} \cdots \sum_{RV(e_1\theta_m)} \cdots \sum_{RV(e_n\theta_m)} \big(\prod_{\theta' \in [C_g\theta_1]} \phi(A_g\theta_1\theta') \big) \ldots \big(\prod_{\theta' \in [C_g\theta_m]} \phi(A_g\theta_m\theta') \big) \\
&= \Big(\sum_{RV(e_1\theta_1)} \cdots \sum_{RV(e_n\theta_1)} \prod_{\theta' \in [C_g\theta_1]} \phi(A_g\theta_1\theta') \Big) \ldots \Big(\sum_{RV(e_1\theta_m)} \cdots \sum_{RV(e_n\theta_m)} \prod_{\theta' \in [C_g\theta_m]} \phi(A_g\theta_m\theta') \Big) \\
&= \prod_{\theta \in [C_{g|L}]} \sum_{RV(e_1\theta)} \cdots \sum_{RV(e_n\theta)} \Big(\prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \Big) \\
&= \prod_{\theta \in [C_{g|L}]} \sum_{RV(E\theta)} \Big(\prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \Big) \\
&= \prod_{C \in U_L(C_g)} \prod_{\theta \in [C]} \sum_{RV(E\theta)} \Phi(g\theta) = \prod_{C \in U_L(C_g)} \prod_{\theta \in [C]} \Phi(g_C\theta) = \prod_{C \in U_L(C_g)} \Phi(g_C).
\end{aligned}
$$

Condition 3.2 is used to ensure the summations on $\sum_{RV(e_1\theta_1)} \cdots \sum_{RV(e_n\theta_m)}$ are indeed distinct. It implies $\forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(E\theta_i) \cap RV(E\theta_j) = \emptyset$, which ensures that the innermost products are on distinct sets of random variables and can therefore be factored out as shown. □

Note that condition 3.2 is equivalent to

$$
\begin{aligned}
&\forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow \neg \exists \theta \cdot e_k\theta_i\theta = e_l\theta_j\theta \\
&\Leftrightarrow \forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i = \theta_j \vee \neg \exists \theta \cdot e_k\theta_i\theta = e_l\theta_j\theta \\
&\Leftrightarrow \forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \neg(\theta_i \neq \theta_j \wedge \exists \theta \cdot e_k\theta_i\theta = e_l\theta_j\theta) \\
&\Leftrightarrow \forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \neg(\exists \theta \cdot \theta_i \neq \theta_j \wedge e_k\theta_i\theta = e_l\theta_j\theta) \\
&\Leftrightarrow \forall e_k, e_l \in E \cdot \neg(\exists \theta_i, \theta_j \in [C_{g|L}] \cdot \exists \theta \cdot \theta_i \neq \theta_j \wedge e_k\theta_i\theta = e_l\theta_j\theta) \\
&\Leftrightarrow \forall e_k, e_l \in E \cdot \neg(\exists L_1, L_2 \cdot C_g\theta_{1|L_1} \wedge C_g\theta_{2|L_1} \wedge \exists \theta \cdot L_1 \neq L_2 \wedge e_k\theta_1\theta = e_l\theta_2\theta) \\
&\Leftrightarrow \forall e_k, e_l \in E \cdot \neg(\exists L_1, L_2 \cdot \exists \theta \cdot C_g\theta_{1|L_1} \wedge C_g\theta_{2|L_1} \wedge L_1 \neq L_2 \wedge e_k\theta_1\theta = e_l\theta_2\theta)
\end{aligned}
$$

where θ_1 and θ_2 are one-to-one mappings from L to standardized apart tuples of logical variables L_1 and L_2

respectively. This is equivalent to the formula

$$C_g\theta_{1|L_1} \wedge C_g\theta_{2|L_1} \wedge L_1 \neq L_2 \wedge e_k\theta_1 = e_l\theta_2$$

being not satisfiable, for all atoms $e_k, e_l \in E$, which the constraint solver can decide.

3.3.3 Propositionalization

When no lifted operation can be applied in order to calculate $\sum_{RV(E)} \Phi(g)$, one can resort to regular propositionalization. In this case, a regular propositional algorithm is applied to the propositional graphical model formed by the set of all instantiations of g. In fact, if g is the fusion of a set of parfactors G_E, it is better to instantiate the parfactors in G_E directly (that is, not to use the fused parfactor), since this will keep them separated and better represent independences in the model, which can be used by the algorithm for greater efficiency.

3.3.4 The Algorithm

Figure 3.6 shows the main pseudocode for FOVE. The algorithm consists of successively choosing eliminations $(E, \{L_1, \ldots, L_k\})$, consisting of a collection of atoms E to eliminate after performing a series of inversions based on sets $L_1, \ldots, L_k$ of logical variables. A possible way of choosing eliminations is presented in figure 3.7. It is presented separately from the main algorithm for clarity, but because these two phases have many operations in common, actual implementations will typically integrate them more tightly.

There are potentially many ways to choose eliminations. The one we present starts by choosing an atom and checking if its inversion will produce a propositional summation, since this is the most efficient case. If not, we successively add atoms to E until G_E forms a parfactor where all atoms with logical variables are part of E (because counting elimination requires it). Then, for efficiency and to avoid shared logical variables between atoms, we try to determine as many inversions as possible, coded in the sequence $L_1, \ldots, L_k$, to be done before counting elimination (or explicit summation when counting cannot be done).

A complexity analysis of FOVE is provided in appendix E.

3.4 An Empirical Example

We use the implementation available at `http://l2r.cs.uiuc.edu/~cogcomp` to compare average run times between lifted and propositional inference (which produce the exact same results) for two different models

FUNCTION *FOVE*(G, Q)
G a set of parfactors, $Q \subseteq RV(G)$, G shattered against Q (section 3.6.2).
1. If $RV(G) = Q$, return G.
2. $(E, \{L_1, \ldots, L_k\}) \leftarrow$ *CHOOSE-ELIMINATION*(G, Q).
3. $g_E \leftarrow fs(G_E)$ (fusion, section 3.6.1).
4. $G' \leftarrow$ *ELIMINATE*(g_E, E).
5. Return *FOVE*$(G' \cup G_{\neg E}, Q)$.

FUNCTION *ELIMINATE*$(g, E, \{L_1, \ldots, L_k\})$
1. If $k = 0$ (no inversion)
   return *SUMMATION-WITHOUT-INVERSION*(g, E).
2. $E_1 \leftarrow \{e \in E : LV(e) \cap L_1 \neq \emptyset\}$ (get inverted atoms).
3. Return $\bigcup_{C_1 \in U_L(C_g)}$ *ELIMINATE-GIVEN-UNIFORMITY*$(g, E_1, C_1, \{L_2, \ldots, L_k\})$.

FUNCTION *ELIMINATE-GIVEN-UNIFORMITY*$(g, E_1, C_1, \{L_2, \ldots, L_k\})$
1. Choose $\theta_1 \in [C_1]$ (bind inverted logical variables arbitrarily).
2. $G' \leftarrow$ *ELIMINATE*$(g\theta_1, E_1\theta_1, \{L_2, \ldots, L_k\})$.
3. $G'' \leftarrow \bigcup_{g'\theta_1 \in G'} (\phi_{g'}, A_{g'}, C_1 \wedge C_{g'})$.
4. Return $\bigcup_{g'' \in G''}$ *SIMPLIFICATION*(g'') (simplification, section 3.6.3).

FUNCTION *SUMMATION-WITHOUT-INVERSION*(g, E)
1. If $E = \{E_1, \ldots, E_k\}$ atoms are independent given C_g and $A_g \setminus E$ is ground
   return $(\sum_{\vec{N}_E} \vec{N}_E! \prod_v \phi_g(v, A_g \setminus E)^{\#(v, \vec{N}_E)}, A_g \setminus E, \top)$ (counting, section 3.3.1).
2. Return $(\sum_{RV(E)} \prod_{\theta \in \Theta_g} \phi_g(A_g\theta_g), A_g \setminus E, C_g)$ (propositional elimination).

Notation:
- $LV(\alpha)$: logical variables in object α.
- $g\theta$: parfactor $(\phi_g, A_g\theta, C_g\theta)$.
- $U_L(C_g)$: USCP of L with respect to C_g (section 3.3.2).
- $C_{|L}$: constraints projected to a set of logical variables L.
- G_E: subset of parfactors G which depend on $RV(E)$.
- $G_{\neg E}$: subset of parfactors G which do not depend on $RV(E)$.
- $\#(v, \vec{N}_E) = \#(v, \vec{N}_E) = \prod_{i=1}^{k} \vec{N}_{E_i, v_i}$
- $\top$: tautology constraint.

Figure 3.6: The FOVE algorithm.

while increasing the number of objects in the domain. The first one, (I) in figure 3.8, answers the query $P(p)$ from a parfactor on $(p, q(X))$ and uses inversion elimination only. The inference in (II) answers query $P(r)$ from a parfactor on $(p(X), p(Y), r)$ and uses counting elimination only. In both cases propositional inference starts taking very long before any noticeable variation in lifted inference run times.

3.5 Applicability of Lifted Inference

As explained in the previous sections, the lifted operations of FOVE are not always applicable, each of them requiring certain conditions to be satisfied in advance. Therefore a natural question is to what kinds of FOPMs we can apply FOVE in an exclusively lifted manner.

It is not clear at this point whether it is possible to tell in advance if a FOPM can be solved with lifted

FUNCTION *CHOOSE-ELIMINATION*(G, Q)
1. Choose e from $A_G \setminus Q$.
2. $g \leftarrow fs(G_e)$ (fusion, section 3.6.1).
3. If $LV(e) = LV(g)$ and $\forall e' \in A_g\ RV(e') \neq RV(e)$
   return $(\{e\}, LV(e))$ (inversion eliminable).
4. $E \leftarrow \{e\}$.
5. While $E \neq$ non-ground atoms of G_E
   $E \leftarrow E \cup$ non-ground atoms of G_E.
6. Return $(E,$ *GET-SEQUENCE-OF-INVERSIONS*$(fs(G_E)))$.

FUNCTION *GET-SEQUENCE-OF-INVERSIONS*(g)
1. If there is no L_1 set of invertible logical variables in g (inversion, section 3.3.2)
   return $\emptyset$.
2. Choose $\theta_1 \in [C_{g|L_1}]$.
3. $\{L2, \ldots, L_k\} \leftarrow$ *GET-SEQUENCE-OF-INVERSIONS*$(g\theta_1)$.
4. Return $\{L_1, L_2, \ldots, L_k\}$.

Figure 3.7: One possible way of choosing an elimination.

operations alone. The main reason for this is that lifted operations will be applied to parfactors resulting from previous operations, so we do not know them in advance. It may be that two parfactors satisfying the lifted operations conditions fuse to form one which does not.

It is possible, however, to run the algorithm symbolically, without performing the actual numerical computations, in order to determine the time they will take. In fact, this same method can be used in order to choose the best elimination ordering. This is analogous to performing VE in propositional graphical models without computing the actual numerical values so as to determine the best elimination ordering.

As a summarization, the conditions for applying lifted operations to eliminate a set $RV(E)$ from a parfactor g are the following: for counting elimination, the atoms in g must be independent given its C_g; for inversion on $L \subseteq LV(g)$,

$$\forall e_k, e_l \in E \cdot \forall \theta_i, \theta_j \in [C_{g|L}] \cdot \theta_i \neq \theta_j \Rightarrow RV(e_k\theta_i) \cap RV(e_l\theta_j) = \emptyset.$$

When lifted operations do not apply, FOVE uses non-lifted operations to calculate $\sum_{RV(E)} G_E$. These non-lifted methods could be propositionalization, sampling etc, but with the advantage of being restricted to a subset of the model only.

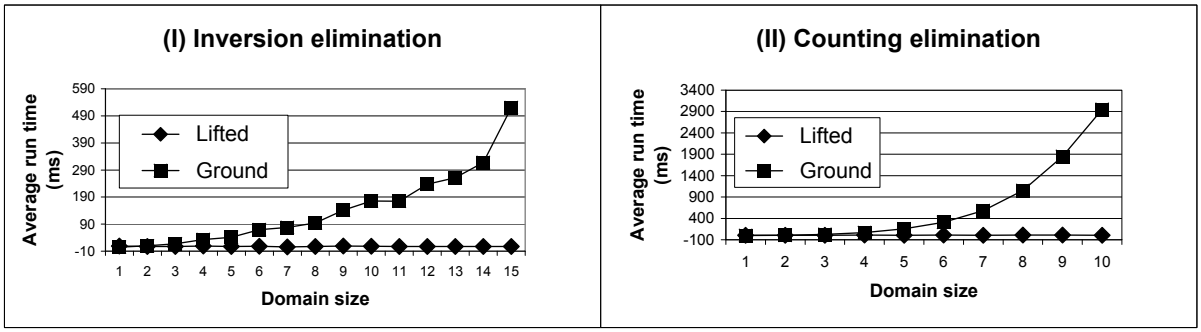

Figure 3.8: (I) Average run time for answering query $P(p)$ from a parfactor on $(p, q(X))$, using inversion elimination, with domain size $|X|$ being gradually increased. (II) Average run time for answering query $P(r)$ from a parfactor on $(p(X), p(Y), r)$, using counting elimination, with domain size $|X| = |Y|$ being gradually increased.

3.6 Auxiliary Operations

3.6.1 Fusion

We have assumed in section 3.3 that we have operations to calculate $\sum_{RV(E)} \Phi(G_E)$, but elimination operations calculate $\sum_{RV(E)} \Phi(g)$, for g a *single* parfactor. *Fusion* bridges this gap by computing, for any set of parfactors G, a single parfactor $fs(G)$ such that $\Phi(G) = \Phi(fs(G))$.

Fusion works by replacing the constraints of all parfactor in the set by a single, common constraint which is the conjunction of them all. This guarantees that all parfactors get instantiated by the same set of substitutions on a single set of logical variables, which allows their products (in the expression for $\Phi(G)$) to be unified under a single product. Note that not all parfactors contain all the logical variables, and will be instantiated to the same ground factor by distinct substitutions (those agreeing on the logical variables present in the parfactor, but disagreeing on some of the others). In other words, some of the parfactors will have their number of instantiations increased by this unification. For this reason, we also exponentiate the potential function to the inverse of how many times the number of instantiations was increased, keeping the final result the same as before.

This is illustrated in the example below:

$$\begin{aligned}\Big(\prod_{D,P} \phi_1(e(D), s(D,P))\Big)\Big(\prod_{D'} \phi_2(e(D'))\Big) &= \prod_{D,P,D'} \phi_1(e(D), s(D,P))\phi_2^{\frac{1}{|D,P|}}(e(D')) \\ &= \prod_{D,P,D'} \phi_3(e(D), s(D,P), e(D')).\end{aligned}$$

(note that logical variables in different parfactors must be standardized apart.) Formally, we have the fusion theorem below.

Theorem 3 (Fusion)**.** Let G be a set of parfactors. Define $C_G = \bigwedge_{g \in G} C_g$, $\Theta_G = [C_G]$ and $A_G = \bigcup_{g \in G} A_g$. Let $fs(G)$ be the parfactor $(\prod_{g \in G} \phi_g^{|\Theta_g|/|\Theta_G|}, A_G, C_G)$. Then $\Phi(G) = \Phi(fs(G))$.

Proof.

$$\begin{aligned}\Phi(G) &= \prod_{g \in G} \prod_{\theta \in \Theta_g} \phi_g(A_g\theta) = \prod_{g \in G} \prod_{\theta \in \Theta_G} \phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|} \\ &= \prod_{\theta \in \Theta_G} \prod_{g \in G} \phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|} = \prod_{\theta \in \Theta_G} \phi_{fs(G)}(A_G\theta) = \Phi(fs(G))\end{aligned}$$

□

While the above is correct, it is rather unnatural to have $e(D)$ and $e(D')$ be distinct atoms. If a set of logical variables has the same possible substitutions, like D and D' here, we can do something better:

$$\begin{aligned}\Big(\prod_{D,P} \phi_1(e(D), s(D,P))\Big)\Big(\prod_{D'} \phi_2(e(D'))\Big) &= \Big(\prod_{D} \prod_{P} \phi_1(e(D), s(D,P))\Big)\Big(\prod_{D'} \phi_2(e(D'))\Big) \\ &= \prod_{D''}\Big((\prod_{P} \phi_1(e(D''), s(D'',P)))\big(\phi_2(e(D''))\big)\Big) \\ &= \prod_{D''}\Big(\prod_{P} \phi_1(e(D''), s(D'',P))\phi_2(e(D''))^{\frac{1}{|P|}}\Big) \\ &= \prod_{D'',P} \phi_1(e(D''), s(D'',P))\phi_2^{\frac{1}{|P|}}(e(D'')) \\ &= \prod_{D'',P} \phi_3(e(D''), s(D'',P)).\end{aligned}$$

Formally, this process is similar to Inversion with respect to D''. However, it does require the additional previous step of unifying distinct logical variables (but with identical sets of possible substitutions) into a single one first (in the example, D and D' are replaced by D''). For lack of space we omit the details of this improvement.

A more complete treatment of fusion is given in appendix B.

3.6.2 Shattering

In section 3.3 we mentioned the need for *shattering*, which we now discuss in more detail. This need arises from c-atoms representing overlapping, but not identical, classes of random variables. Consider the following

marginalization over parfactors g_1 and g_2 with potential functions ϕ_1 and ϕ_2 respectively:

$$\sum_{RV(p(X,Y))} \Big(\prod_{X,Y} \phi_1(p(X,Y),q) \Big) \prod_{Y} \phi_2(p(a,Y))$$

If we pick $E = p(a,Y)$, $G_E = \{g_1, g_2\}$. However, only some instantiations of g_1 depend on $p(a,Y)$ (the ones with $X = a$). Moreover, the operations we later talk about require any pair of c-atoms in G_E to represent either identical classes of random variables, or those classes to be disjoint. This is violated by $RV(p(a,Y))$ being a subset of $RV(p(X,Y))$. Picking $E = p(X,Y)$ also violates this requirement.

The solution is to **split** parfactor g_1 into two different parfactors. The union of instantiations of $(\phi_1, (p(X,Y),q), X \neq a)$ and $(\phi_1, (p(a,Y),q), \top)$ is identical to the set of instantiations of g_1, so the summation can be simply rewritten as

$$\begin{aligned}
&\sum_{RV(p(X,Y))} \Big(\prod_{X,Y:X\neq a} \phi_1(p(X,Y),q) \Big) \Big(\prod_{Y} \phi_1(p(a,Y),q) \Big) \prod_{Y} \phi_2(p(a,Y)) \\
&= \sum_{RV(p(X,Y):X\neq a)} \Big(\prod_{X,Y:X\neq a} \phi_1(p(X,Y),q) \Big) \sum_{p(a,Y)} \Big(\prod_{Y} \phi_1(p(a,Y),q) \Big) \prod_{Y} \phi_2(p(a,Y))
\end{aligned}$$

Now $E = p(a,Y)$ satisfies the operations' requirements. Picking $E = p(X,Y), X \neq a$ would work equally well.

Splitting parfactors is done by pairwise comparisons of atoms of same predicate. We split parfactors $g_1 = (\phi_1, A_1, C_1)$ and $g_2 = (\phi_2, A_2, C_2)$ around atoms $a_1 \in A_1$ and $a_2 \in A_2$ by replacing them by parfactors $(\phi_1, A_1, C_1 \wedge a_1 = a_2)$, $(\phi_1, A_1, C_1 \wedge a_1 \neq a_2)$, $(\phi_2, A_2, C_2 \wedge a_1 = a_2)$ and $(\phi_2, A_2, C_2 \wedge a_1 \neq a_2)$, after standardizing apart their logical variables. In fact, we only need to keep those whose constraint is satisfiable. (This is why g_2 does not need to be broken in the example above – that would only produce itself and another parfactor with zero instantiations.) In particular, if $RV(a_1) = RV(a_2)$, we end up obtaining the original parfactors.

The uniformity requirement is met by **shattering** the FOPM in advance, that is, by successively splitting the parfactors of each pair of c-atoms, including the query atoms, until no overlapping non-identically grounded pair remains. The query atoms need to be involved in shattering because if a c-atom includes query and non-query random variables, it needs to be split so that the non-query ones can be eliminated.

As pointed out by [Poo03], splitting parfactors resembles the role of unification in first-order resolution, which determines the conditions for two atoms to match. In probabilistic inference, however, we are interested

not only in the overlapping of atoms but also in the *residual* parfactors that originate from the matching. The reason for this difference is that the *number* of instantiations of a parfactor matters for the final joint distribution. In regular resolution, the original clauses are kept because their redundancy with the clauses resulting from resolution makes no difference, while here we need to discount them and replace the originals with the non-matching cases.

A more complete treatment of shattering is given in appendix C.

3.6.3 Irrelevant Logical Variable Simplification

Inversion often produces parfactors with constraints with logical variables not present in its atoms. The first Inversion example produces the expression below. We can simplify it by observing that the actual value of Y is irrelevant inside the product. Only the number $|Y|$ of possible values for Y will make a difference. Therefore we can write

$$\prod_{XY} \phi'(p(X)) = \prod_{X} \phi'(p(X))^{|Y|} = \prod_{X} \phi''(p(X)).$$

Chapter 4

Lifted Most Probable Explanation

The previous chapter introduced FOVE for solving the problem of marginalization of certain random variables in a FOPI model. Marginalization is arguably the most important probabilistic inference problem, but another important one is that of finding the Most Probably Explanation (MPE) [Pea88]. In this problem, we seek the assignment to the random variables in a model with the maximum probability. This chapter explains what is involved in solving the same problem for FOPI models.

4.1 Lifted Assignments

In propositional probabilistic models, assignments to random variables can be seen as formulas with equality on those random variables. For example, $alarm = false \wedge earthquake = true$ can be regarded as an assignment. The formal link between assignment and formula is that the former satisfies the latter. A total assignment (that is, one with a value for each and all random variables in the model) can be represented by a formula which is satisfied only by that total assignment.

While the same is true for first-order models, that is, we can represent individual assignments by formulas that test the value for each random variable in the model, we are often interested in *sets* of assignments with identical behavior with respect to the model. The reason is that first-order models may contain classes of *interchangeable* random variables. For example, in a model defined by a single parfactor $\phi(epidemic, sick(P))$, the assignment $epidemic = false, sick(john) = true, sick(P) = false$ for $P \neq john$ and the assignment $epidemic = false, sick(mary) = true, sick(P) = false$ for $P \neq mary$ are equally likely because all random variables instantiated from $sick(P)$ are interchangeable. In fact, any pair of assignments to $RV(sick(P))$ with the same number of $true$ and $false$ values will be equally likely. What matters is the distribution of values assigned to $RV(sick(P))$, rather than the individual assignments. This distribution corresponds to the concept of a counter $\vec{N}_{sick(P)}$ in counting elimination.

Therefore, we define the notion of *lifted assignments* in first-order models. Lifted assignments correspond to sets of interchangeable assignments, that is, assignments that only vary with respect to interchangeable

random variables. As with regular propositional assignments, lifted assignments can also be represented by a formula with equality. These formulas, however, are first-order and can be quantified. In order to quantify distributions over sets of exchangeable random variables, we introduce a **special existential quantifier**. Its general form is $[\exists^{n(v)}_{v \in D}\ L : C]\ \varphi(v)$, which means that, for every $v \in D$, there are $n(v)$ substitutions for logical variables L satisfying constraint C for which $\varphi(v)$ holds. When the constraint is the most general $\top$, we simply write $\exists^{n(v)}_{v \in D}\ L\ \varphi(v)$.

An example of this existential quantifier is $[\exists^{n(v)}_{v \in \{false,true\}} P : P \neq julie]\ sick(P)$, for $n(true) = 57$ and $n(false) = 42$ (we assume the domain of P to have size 100, including $julie$), which declares that 57 random variables in $RV(sick(P)), P \neq julie$ are assigned $true$ and 42 are assigned $false$.

For the particular case where D contains a single value v and $n(v) = |[C]|$, we simply write $[\forall L : C]\ \varphi$, or $\forall L\ \varphi$ when $C = \top$. We call it a **universal quantifier**.

4.2 The mpe Operator

In the Most Probable Explanation inference problem (MPE) we must identify some assignment to random variables with maximum potential:

$$\text{Maximum Potential of G} = \max_{RV(G)} \Phi(G)$$

The expression above is very similar to $\sum_{RV(G)} \Phi(G)$. In *propositional* probabilistic inference, a common way of solving this problem is to use Variable Elimination, but replacing summations by maximizations. The maximizing assignment can then be trivially obtained as a side effect of maximizations, although it is not explicitly represented in the expression above.

The solution to the MPE problem is the assignment generating the above maximum potential:

$$\text{MPE of G} = \arg\max_{RV(G)} \Phi(G)$$

Here, however, one cannot simply replace $\sum$ by $\arg\max$ in the algorithm. The reason for that is there is a nesting of summations. An inner summation results in a real number representing a potential, which is also the argument to the next summation. The operator $\arg\max$, on the other hand, has a potential as an argument but results in an assignment, which cannot be the argument to the next $\arg\max$. In other words,

one can write

$$\text{Marginal of G} = \sum_{RV(G)} \Phi(G) = \sum_{r_1} \sum_{r_2} \cdots \sum_{r_n} \Phi(G)$$

where $RV(G) = \{r_1, \ldots, r_n\}$ but not

$$\text{MPE of G} = \arg\max_{RV(G)} \Phi(G) = \arg\max_{r_1} \arg\max_{r_2} \ldots \arg\max_{r_n} \Phi(G) \text{ (\textbf{undefined!})}$$

The reason behind this problem is that MPE has to consider the potentials while keeping track of the assignments maximizing them. To solve this, we introduce a new operator that manipulates *pairs* of potentials and assignments, the mpe operator.

For any real function f and variable q, we define the **application of** mpe **to a real-valued function** as

$$\operatorname*{mpe}_q f(q) = \Big(\max_{q'} f(q'), q = \arg\max_{q'} f(q')\Big).$$

The result of mpe is a pair of the maximum value of $f(q)$ and the maximizing assignment to q, represented by a formula with equality. We call such a pair p a *potential-assignment pair*, or pa-pair, and its components are $p^{\mathbf{P}}$ and $p^{\mathbf{A}}$ respectively. Let $\phi(q)$ be a function returning potential-assignment pairs. Then we define the **application of** mpe **to a pa-pair-valued function** as

$$\operatorname*{mpe}_q \phi(q) = \Big(\max_{q'} \phi^{\mathbf{P}}(q'), [q = \arg\max_{q'} \phi^{\mathbf{P}}(q')] \wedge \phi^{\mathbf{A}}(q)\Big),$$

that is, if given a function that provides pa-pairs, mpe returns a pair maximizing that potential for q, with an assignment represented by the conjunction of maximizing assignment to q and whatever assignment $\phi^{\mathbf{A}}(q)$ is.

Note how the mpe operator has the effect of eliminating a variable. The first component of the resulting pa-pair does not contain the variable anymore, since it maximizes it away. The second component records the maximizing argument, while keeping whatever other records of previous eliminations there were, by conjunction. The effect we obtain is that, as the mpe operator is applied, we maximize variables away, while recording the maximizing assignments in the second component.

The mpe operator can then replace $\sum$ in the VE and FOVE algorithms, since we can write

$$\text{MPE of G} = \max_{RV(G)} \Phi(G) = \max_{r_1} \max_{r_2} \ldots \max_{r_n} \Phi(G)$$

4.3 FOVE for MPE

Once we have the mpe operator and parfactors that map to pa-pairs rather than simply potentials, we can use the FOVE algorithm to solve MPE by replacing $\sum$ by mpe and having an empty query (so that the maximization occurs over all random variables). The use of mpe guarantees that maximization is being performed and that elimination operations produce parfactors returning maximizing pa-pairs with assignments to previously eliminated variables.

On inspecting the FOVE algorithm, however, one can see that summation is not the only operation performed on potentials and potential functions. Potentials are also multiplied by other potentials, and repeated multiplication gives rise to exponentiation at points. Also, for counting elimination, we convert potential functions having value tuples as arguments (parfactors' potential functions) to potential functions having counters as arguments. Therefore, in order to run the algorithm on pa-pairs and pa-pairs functions, we need to show how to perform products on pa-pairs, as well as how to convert pa-pair functions on value tuples to pa-pair functions on counters.

4.3.1 Pa-Pair Products

Product over pa-pairs is straightforward. The product of reals is kept for the potential component of the pair, while conjunction is used for the formula component. For r and s pa-pairs, $r \times s = (r^{\mathbf{P}} s^{\mathbf{P}}, r^{\mathbf{A}} \wedge s^{\mathbf{A}})$.

The pa-pair product contributes in forming universally quantified lifted assignments because universal quantification is a form of conjunction. Assume that g is a parfactor on $p(X), q(X,Y)$ with marginals given by $\phi^{\mathbf{P}}$ and $\phi^{\mathbf{A}}$ defining assignments on a previously removed $r(X,Y)$ by a function f. We can invert X, Y and obtain (ellipsis $(\ldots)$ is used to avoid repetition of long segments)

$$\begin{aligned}
\max_{q(X,Y)} \Phi(g) &= \max_{q(X,Y)} \prod_{X,Y} \Big(\phi_g^{\mathbf{P}}(p(X), q(X,Y)),\ r(X,Y) = f(p(X), q(X,Y)) \Big) \\
&= \prod_{X,Y} \max_{q(X,Y)} \Big(\phi_g^{\mathbf{P}}(p(X), q(X,Y)),\ r(X,Y) = f(p(X), q(X,Y)) \Big) \\
&= \prod_{X,Y} \Big(\max_{q(X,Y)} \phi_g^{\mathbf{P}}(p(X), q(X,Y)),\ q(X,Y) = \arg\max_{q(X,Y)} \phi_g^{\mathbf{P}}(\ldots) \wedge r(\ldots) = f(\ldots) \Big)
\end{aligned}$$

(by defining appropriate ϕ' and f')

$$= \prod_{X,Y} \Big(\phi'^{\mathbf{P}}(p(X)),\ q(X,Y) = f'(p(X)) \wedge r(\dots) = f(\dots) \Big)$$

(because the marginal depends only on $p(X)$ and the pa-pair product results in the conjunction of lifted assignments)

$$= \prod_{X} \Big(\phi'^{\mathbf{P}}(p(X))^{|Y|},\ \forall Y\ q(X,Y) = f'(p(X)) \wedge r(\dots) = f(\dots) \Big)$$
$$= \Phi(g')$$

for some appropriate parfactor $g' = (\phi_{g'}, \{p(X)\}, C_{g'})$.

The interpretation of this lifted assignment is that, for the X in question, it is true that, for every Y, $q(X,Y)$ is assigned $f'(v)$ and $r(X,Y)$ is assigned $f(v, q(X,Y)) = f(v, f'(v))$.

In general, universally quantified lifted assignments are formed at the point of the algorithm where a parfactor g has an irrelevant subset L of logical variables so that we have

$$\prod_{\theta_L \in [C_{g|L}]} \prod_{\theta \in [C_{g|LV(g)\setminus L}]} \Big(\phi^{\mathbf{P}}(A_g\theta), \phi^{\mathbf{A}}(A_g\theta) \Big) = \prod_{\theta \in [C_{|LV(g)\setminus L}]} \Big((\phi^{\mathbf{P}})^{|[C_{g|L}]|}(A_g\theta), [\forall L : C_{g|L}] \phi^{\mathbf{A}}(A_g\theta) \Big)$$

4.3.2 Conversion to Potential Functions on Counters

We now show the conversion of pa-pair functions on value tuples to pa-pair functions on counters by example and in general. In the following, mpe is over $RV(p(X))$, which is the same as $RV(p(Y))$:

$$\operatorname*{mpe}_{p(X)} \prod_{X \neq Y} \Big(\phi^{\mathbf{P}}(p(X), p(Y)),\ r(X,Y) = f(p(X), p(Y)) \Big)$$

(by expressing the product as the number of times ϕ is applied to each possible argument v)

$$= \operatorname*{mpe}_{p(X)} \Big(\prod_{v \in D_{p(X),p(Y)}} \phi^{\mathbf{P}}(v)^{\#(v, \vec{N}_{p(X)})}, [\exists_{v \in D_{p(X),p(Y)}}^{\#(v, \vec{N}_{p(X)})} XY : X \neq Y]\ r(X,Y) = f(v) \Big)$$

(by grouping assignments on $RV(p(X))$ by $\vec{N}_{p(X)}$)

$$= \max_{\vec{N}_{p(X)}} \Big(\prod_{v \in D_{p(X),p(Y)}} \phi^{\mathbf{P}}(v)^{\#(v,\vec{N}_{p(X)})}, [\exists^{\#(v,\vec{N}_{p(X)})}_{v \in D_{p(X),p(Y)}} XY : X \neq Y]\ r(X,Y) = f(v) \Big)$$

(by defining appropriate f' and f'')

$$= \max_{\vec{N}_{p(X)}} \Big(f'(\vec{N}_{p(X)}), f''(\vec{N}_{p(X)}) \Big)$$

(by applying the definition of mpe)

$$= \Big(\max_{\vec{N}_{p(X)}} f'(\vec{N}_{p(X)}), \vec{N}_{p(X)} = \arg\max_{\vec{N}_p p(X)} f'(\vec{N}_p p(X)) \wedge f''(\vec{N}_{p(X)}) \Big)$$

(by defining an appropriate parfactor $g' = (\phi', \emptyset, \top)$)

$$= \Big(\phi'^{\mathbf{P}}(), \phi'^{\mathbf{A}}() \Big) = \Phi(g').$$

g' is a constant parfactor returning the maximum potential and the lifted assignment formed by the distribution of values on $p(X)$ (represented by $\vec{N}_{p(X)}$) maximizing f', and, for each $v \in D_{p(X),p(Y)}$, there are $\#(v, \vec{N}_{p(X)})$ pairs (X, Y) satisfying $X \neq Y$ such that $r(X, Y)$ is assigned $f''(v)$.

In general, existentially quantified lifted assignments are formed at the point of the algorithm where a parfactor g is rewritten as a function of counters, so that we have

$$\prod_{\theta \in [C_g]} \Big(\phi_g^{\mathbf{P}}(A_g\theta), \phi_g^{\mathbf{A}}(A_g\theta) \Big) = \Big(\prod_{v \in D_{A_g}} \phi_g^{\mathbf{P}}(v)^{\#(v,\vec{N}_{A_g})}, \exists^{\#(v,\vec{N}_{A_g})}_{v \in D_{A_g}} \phi_g^{\mathbf{A}}(v) \Big)$$

With these operations defined, FOVE can be used to solve the MPE problem.

4.4 Example

We now illustrate MPE inference with partial inversion in a simple business domain. We want to express the fact that some companies establish partnerships, and it will often be the case that, for each product in

question, one of the partners is a retail company while the other is not. We can write

$$\phi_1(partners(P, C_1, C_2)), C_1 \neq C_2.$$
$$\phi_2(partners(P, C_1, C_2), retail(C_1), retail(C_2)), C_1 \neq C_2.$$

where ϕ_1 and ϕ_2 are parfactors with appropriate potential functions for the desired dependencies. ϕ_1 performs the role of a "prior" (although such a notion is not precise in an undirected model as this). If there are 15 companies in the domain, our implementation (available at `http://l2r.cs.uiuc.edu/~cogcomp/`) produces a lifted assignment that can be read as
"for all products, there are 8 retail companies and 7 non-retail companies. 56 pairs of companies are partners and the rest is not".

It is interesting to note that the numbers above do not depend on the particular potentials, but only on the assignments made more likely by each parfactor. This suggests that FOVE can be further developed to take advantage of MPE being an easier problem than Belief Assessment, avoiding the iteration over counters and instead deducing the maximum one from the parfactor structure alone. This would make MPE's time complexity completely independent from the domain size.

Chapter 5

Blanket Bound

An important feature of logical systems is the fact that the model is analyzed during inference only until the answer to a current query is found. Therefore, very large knowledge bases do not have to be thoroughly processed for every query, and simple queries may be answered in relatively short time.

A first-order probabilistic model should do no worse when processing a deterministic knowledge base, that is, it should restrict itself to the relevant part of the model for the current query only. This means, in the case of a VE-like algorithm, not to eliminate all random variables, but only those necessary for answering the query. Besides, almost-deterministic models should also reflect this property with graceful degradation.

One of the main current disadvantages of the FOVE algorithm is the need to shatter the entire model in advance, regardless of the query. A gradual processing of the model, shattering parfactors only as necessary, would be more desirable. To achieve that, however, we need to be able to use a partial evaluation of the model. While a partial evaluation cannot in general provide an answer, it can provide a bounded answer. So we expect to extend FOVE to gradually process and shatter a model and derive bounds as a result.

This chapter presents Blanket Bound, a notion that is a first step towards that goal. We have so far developed it for the propositional case only, with the intention of generalizing it to the first-order case and FOVE.

5.1 Introduction

Graphical models (of which Bayesian and Markov models are examples) represent a joint probability distribution on a set of random variables by representing dependencies between them. This is done by a set of *factors*, or *potential functions*, functions on subsets of variables indicating the relative compatibility of their assignments. The most common inference task in graphical models is calculating the marginal probability of a subset of variables, which can be solved by the Variable Elimination (VE) algorithm (described in [DR03]), among others. VE works by eliminating each of the non-query random variables in a way that preserves marginal probabilities, until a model on the query only is left, and from which its marginal probability is

easily determined. Each variable is eliminated by summing it out from all factors having it as an parameter, obtaining a new factor without it that replaces them.

Graphical models are a probabilistic generalization of deterministic logical propositional clausal theories, since both factor and clause determine dependencies among subsets of variables. VE can be seen as a generalization of resolution, the main inference technique for clausal theories, since resolution takes clauses involving a variable and computes a new clause without it.

Because of the determinism involved in resolution, typically the truth value of a proposition can be determined without considering all clauses in the theory. Consider the model in figure 5.1(a) and a clausal stating that *errands* is true if any of its parents is true. Then if we know that *groceries* is true, resolution will indicate that *errands* is true without ever processing the other variables. This contrasts with VE on a graphical model: consider the almost deterministic case in which *errands* is a noisy-or node with coefficient 0.9 for each parent (that is, its probability is $0, 0.9, 0.99$ and 0.999 for 0, 1, 2 and 3 parents set to *true*, respectively), and that we know that *groceries* is true. Then, even though the model is very similar to the deterministic case, the exact marginal probability of *errands* depends on *all* variables and a probabilistic inference algorithm will have to consider the *entire* model in order to calculate it. So a slight change from a deterministic to a similar probabilistic model makes it necessary to consider the entire model, when only a fraction of it had to be considered before.

Intuitively, this abrupt change seems wasteful and a more graceful degradation would be expected. While such graceful degradation is not possible with respect to an exact solution (since it does depend on the entire model), it is possible with respect to *partial* ones: knowing *groceries* to be true does not solve the entire problem but solves *most* of it, since it determines that the marginal on *errands* cannot be less than 0.9 regardless of other variables. So there is a graceful degradation as a model goes from deterministic to almost deterministic, in the sense that evaluating the relevant part of a model goes from producing an exact solution to an almost exact solution. However, exact probabilistic inference algorithms such as VE cannot take advantage of this because they do not involve approximation in the first place.

In order to derive useful information from the immediate neighborhood of a query, we have developed a way to calculate lower and upper bounds on the marginal probability of a query based on its immediate neighborhood only. This bound, which we call *blanket bound*, reproduces for graphical models the logical inference quality of drawing inferences without examining the entire model. This bound calculation and its properties are the main contributions of this chapter. We see it as an improvement of the technique known as bound propagation [LK03], which also seeks to compute bounds on marginals from their separators from the rest of the network. Our bound is optimal given the separators, and therefore does as well or better than

bound propagation, while being much simpler to compute. We further discuss this relationship in section 5.7.

We also present an algorithm, which we call *Anytime Bounded VE* (ABVE) that exploits this graceful degradation. As opposed to VE, ABVE can gradually eliminate variables and maintain, at any time, lower and upper bounds on the query's marginal probability. For deterministic models, this will result in exact solutions as soon as the truly relevant variables are considered, as opposed to having to eliminate all non-query variables. This makes ABVE a more adequate generalization of logical inference than VE.

ABVE is an *anytime algorithm*, that is, an algorithm that can be interrupted at any point and still return a useful solution. The more time given to an anytime algorithm, the better the solution becomes, thus providing a flexible tradeoff between time and accuracy. The benefits of anytime algorithms have been pointed out by several authors [HSC89, FH94]. The main one is that inference will be useful even if there is insufficient time for computing an exact solution, an important scenario given the complexity of graphical models inference. Moreover, ABVE is a *bounded approximation* algorithm since it returns guaranteed bounds on the solution. These guarantees are important. For example, they allow us to use bounded approximation to solve decision problems exactly, since in order to choose between two different actions we only need to know that the expected utility of one of them is larger than the other, and this can be done with bounds alone.

While ABVE provides the benefits of all anytime bounded approximation algorithms, we believe it goes beyond them in two important aspects due to its mirroring of logical resolution. First, it does not need to have the entire graphical model available in advance. Its evaluation starts at the query's vicinity and expands from there. At any step, it will never have used factors beyond this boundary. This is crucial in applications in which the model is constructed dynamically [WBG92, Mur02b], since this can be quite expensive. Second, it offers a bridge between deterministic resolution and graphical.models inference.

We present the necessary graphical network background in section 5.2, how to calculate the blanket bound on a marginal probability in section 5.3, describe the ABVE algorithm in section 5.4, discuss preliminary empirical evaluation in section 5.5. In section 5.6 we show that bounds vary monotonically as the algorithm progresses. We discuss related work and future directions in sections 5.7 and 5.8.

5.2 Factor Networks

We develop our algorithm for factor networks, a type of graphical model that can represent both Markov and Bayesian networks [Pea88]. A factor network (of which figure 5.1(b) shows an example) is a graph with

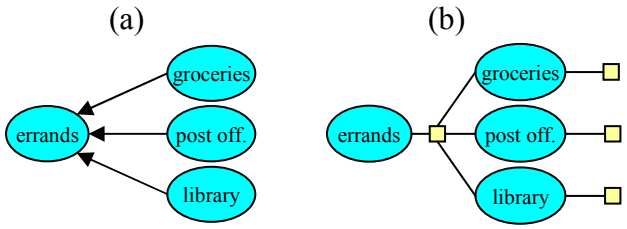

Figure 5.1: (a) Structure of logical or Bayesian models representing a variable *errands* and its multiple causes. (b) A factor network corresponding to the Bayesian network as in (a). The squares represent factors (or potential functions) applied to its neighboring random variables.

two types of nodes: random variable nodes $\mathbf{X} = (X_1, \ldots, X_n)$ and factor nodes $\Phi = \phi_1, \ldots, \phi_m$. Edges are always between a variable node and a factor node. Each variable X has a discrete domain D_X. We define $\Phi_{\mathbf{Y}} \subseteq \Phi$ to be the set of factors neighboring any of the variables in $\mathbf{Y}$, and $\Phi_{\neg \mathbf{Y}}$ to be its complement, and $Var(\Phi)$ to be the random variables which are parameters in the factors of Φ.

An *assignment* is a partial function from random variables to values in their domains. $D_{\mathbf{Y}}$ denotes the set of assignments to a set of variables $\mathbf{Y}$. Assignments are usually denoted as the small letter version of their corresponding variables, or even set expressions; y is an assignment to Y, $\mathbf{y}$ to $\mathbf{Y}$ and $\mathbf{y} \setminus \mathbf{z}$ to $\mathbf{Y} \setminus \mathbf{Z}$. For notation clarity, we assume set difference ($\setminus$) to always be left associative, that is, $\mathbf{x} \setminus \mathbf{y} \setminus \mathbf{q}$ means $(\mathbf{x} \setminus \mathbf{y}) \setminus \mathbf{q}$.

A factor node ϕ is a function on assignments to its neighboring variables $\mathbf{A}_\phi \subset \mathbf{X}$. It maps each assignment $\mathbf{a}_\phi$ on $\mathbf{A}_\phi$ to a non-negative real number $\phi(\mathbf{a}_\phi)$, called the *potential* of $\mathbf{a}_\phi$. The potential represents the degree of compatibility between the values in the assignment.

We extend potential functions in the following way. If $\mathbf{Y} \subseteq \mathbf{X}$, $\phi(\mathbf{y}) = \phi(\mathbf{y}_{|\mathbf{a}_\phi})$, where $\mathbf{y}_{|\mathbf{a}_\phi}$ is the assignment on $\mathbf{Y} \cap \mathbf{A}_\phi$ which agrees with $\mathbf{y}$. That is to say, extra inputs to ϕ are ignored.

For any set of factors Φ, $\Phi(\mathbf{x})$ denotes the function $\prod_{\phi \in \Phi} \phi(\mathbf{x})$.

Although a potential does not necessarily have a probabilistic meaning per se, a factor network defines a joint distribution on its random variables. A joint distribution on $\mathbf{X}$ is defined as $P(\mathbf{x})$ for each assignment $\mathbf{x}$ to $\mathbf{X}$:

$$P(\mathbf{x}) = \frac{1}{Z} \prod_{\phi \in \Phi} \phi(\mathbf{x}) = \frac{1}{Z} \Phi(\mathbf{x})$$

where $Z = \sum_{\mathbf{x}'} \prod_{\phi \in \Phi} \phi(\mathbf{x}') = \sum_{\mathbf{x}'} \Phi(\mathbf{x}')$, a normalization constant. We denote this more simply as

$$P(\mathbf{x}) \propto \Phi(\mathbf{x}).$$

Factor networks can represent both Markov networks (with each factor representing the potential function on a clique of random variables in the network), and Bayesian networks (with each factor representing a conditional probability). Figure 5.1(b) shows a factor network equivalent to a Bayesian network implementing the *errands* example.

5.2.1 Inference

The most important inference problem in graphical models is that of calculating the marginal probability of a set of random variables $\mathbf{Q}$. This is calculated as

$$P(\mathbf{q}) \propto \sum_{\mathbf{x}\setminus\mathbf{q}} \Phi(\mathbf{q}, \mathbf{x} \setminus \mathbf{q})$$

Calculating this summation directly is exponential on the number of variables and therefore intractable. Variable Elimination (VE) (described in [DR03]) seeks to compute it in a more efficient manner. It works by eliminating each non-query variable $Y \in \mathbf{X} \setminus \mathbf{Q}$ in the following way, where $\mathbf{Y}$ denotes the singleton $\{Y\}$:

$$\begin{aligned} P(\mathbf{q}) \propto \sum_{\mathbf{x}\setminus\mathbf{q}} \Phi(\mathbf{x}) &= \sum_{\mathbf{x}\setminus\mathbf{y}\setminus\mathbf{q}} \Phi_{\neg\mathbf{Y}}(\mathbf{x} \setminus \mathbf{y}) \sum_{\mathbf{y}} \Phi_{\mathbf{Y}}(\mathbf{x} \setminus \mathbf{y}, \mathbf{y}) \\ &= \sum_{\mathbf{x}\setminus\mathbf{y}\setminus\mathbf{q}} \Phi_{\neg\mathbf{Y}}(\mathbf{x} \setminus \mathbf{y})\phi(\mathbf{x} \setminus \mathbf{y}) \\ &= \sum_{\mathbf{x}\setminus\mathbf{y}\setminus\mathbf{q}} (\Phi_{\neg\mathbf{Y}} \cup \{\phi\})(\mathbf{x} \setminus \mathbf{y}) = \sum_{\mathbf{x}\setminus\mathbf{y}\setminus\mathbf{q}} \Phi'(\mathbf{x} \setminus \mathbf{y}), \end{aligned}$$

where Φ' does not involve $\mathbf{Y}$ as a parameter, so the final expression above does not involve $\mathbf{Y}$. By repeating this process for all non-query variables, we end up with a model providing the query's marginal probability. This will typically be much cheaper than the original summation (depending on the network's connectivity).

5.3 The Blanket Bound on Marginal Probability

From the above description of VE it is clear that we only obtain a solution after all non-query variables are eliminated. Depending on the model size, this may take too long for some applications, even those in which a bound of the marginal probability would be enough or at least helpful, as illustrated in the *errands* example in the introduction.

We now present a method for deriving lower and upper bounds on $P(\mathbf{q})$ based solely on $\Phi_{\mathbf{Q}}$, the immediate neighboring factors of $\mathbf{Q}$. We first need some definitions: let $\mathbb{P}(\mathbf{Y})$ be the set of possible probability

distributions on any $\mathbf{Y} \subseteq \mathbf{X}$ and $\mathbf{NQ}$ (the *neighboring variables* of $\mathbf{Q}$) the set of variables which occur in $\Phi_{\mathbf{Q}}$ but are not part of $\mathbf{Q}$. Note that $\mathbf{NQ}$ is the Markov blanket of $\mathbf{Q}$, and for this reason we can this bound the *blanket bound*.

Consider the following:

$$\begin{aligned} P(\mathbf{q}) &\propto \sum_{\mathbf{x}\setminus\mathbf{q}} \Phi(\mathbf{x}) = \sum_{\mathbf{nq}} \sum_{\mathbf{x}\setminus\mathbf{q}\setminus\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q},\mathbf{nq})\ \Phi_{\neg\mathbf{Q}}(\mathbf{x}\setminus\mathbf{q}) \\ &= \sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q},\mathbf{nq}) \sum_{\mathbf{x}\setminus\mathbf{q}\setminus\mathbf{nq}} \Phi_{\neg\mathbf{Q}}(\mathbf{x}\setminus\mathbf{q}) \\ &= \sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q},\mathbf{nq})\ \phi(\mathbf{nq}) \\ &\propto \sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q},\mathbf{nq})\ \ \mathbf{p}(\mathbf{nq}) \quad (\mathbf{p} \in \mathbb{P}(\mathbf{NQ})) \end{aligned}$$

More explicitly,

$$P(\mathbf{q}) = \frac{\sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q},\mathbf{nq})\ \ \mathbf{p}(\mathbf{nq})}{\sum_{\mathbf{q}'} \sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q}',\mathbf{nq})\ \ \mathbf{p}(\mathbf{nq})} = f(\mathbf{p}),$$

which is to say that $P(\mathbf{q})$ is a function on $\mathbf{p}$, a probability distribution on $\mathbf{NQ}$. Note that this function depends only on summations over $\mathbf{Q}$ and $\mathbf{NQ}$, and on factors in $\Phi_{\mathbf{Q}}$ so, given $\mathbf{p}$, it is much more tractable than the full marginalization calculation. However, we do not know what $\mathbf{p}$ is, and its value depends on the much more expensive summation on $\Phi_{\neg\mathbf{Q}}$.

Suppose for a moment that we could sidestep this expensive summation and instead directly range over all the infinite possible values of $\mathbf{p}$. This would allow us to observe how $P(\mathbf{q})$ varies as a result and obtain lower and upper bounds for it. Even without knowing what the true value of $\mathbf{p}$ is, we would already have a measure of information on $P(\mathbf{q})$.

Fortunately, this method is actually possible because f is a quotient of linear functions on $\mathbf{p}$, and therefore monotonic on $\mathbf{p}$, and because $\mathbf{p}$ is a probability distribution with clear extremes. As a result, the extrema of $P(\mathbf{q})$ will necessarily correspond to the extrema of $\mathbf{p}$ itself, of which there is a finite number only.

Let us consider this situation graphically. If $\mathbf{NQ}$ has n possible assignments, $\mathbb{P}(\mathbf{NQ})$ forms a convex polytope of dimension $n-1$ embedded in $\mathbb{R}^n$. This polytope has n vertices, each of them corresponding to the distribution in which a certain assignment has probability 1. Figure 5.2 shows $\mathbb{P}(\mathbf{NQ})$ for $\mathbf{NQ}$ with three assignments.

When $\mathbf{nq}$ is a vertex of $\mathbf{p}$, $\mathbf{p}(\mathbf{nq}) = 1$ and $\mathbf{p}(\mathbf{nq}') = 0$ for assignments $\mathbf{nq}' \neq \mathbf{nq}$. This simplifies the

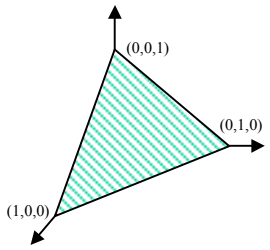

Figure 5.2: $\mathbb{P}(\mathbf{NQ})$ for $\mathbf{NQ}$ with three assignments.

calculation of $P(\mathbf{q})$:

$$P(\mathbf{q}) = \frac{\sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q}, \mathbf{nq}) \;\; \mathbf{p}(\mathbf{nq})}{\sum_{\mathbf{q}'} \sum_{\mathbf{nq}} \Phi_{\mathbf{Q}}(\mathbf{q}', \mathbf{nq}) \;\; \mathbf{p}(\mathbf{nq})} = \frac{\Phi_{\mathbf{Q}}(\mathbf{q}, \mathbf{nq})}{\sum_{\mathbf{q}'} \Phi_{\mathbf{Q}}(\mathbf{q}', \mathbf{nq})}$$

It is important to note that these are the best possible bounds based on $\Phi_{\mathbf{Q}}$, since we can, for each of the lower and upper bounds, build a model for which the marginal will be *equal* to it.

5.3.1 Performing Bound Propagation

If we already have bounds $P^-(\mathbf{nq})$ and $P^+(\mathbf{nq})$ over $\mathbf{NQ}$, they correspond to the vertices of an inner polytope which is a subset of $\mathbb{P}(\mathbf{NQ})$. In this case we can use the function f to calculate bounds on $\mathbf{Q}$ that take these bounds into account:

$$P^+(\mathbf{q}) = \max_{\mathbf{p} \in V} f(\mathbf{p})$$
$$V = \{P^s(\mathbf{nq}) : \mathbf{nq} \in D_{\mathbf{NQ}}, s \in \{-, +\}\}$$

This allows us to perform bound propagation in exactly the same manner as [LK03], but with much simpler computations, since their method requires constructing and solving a linear programming problem.

Determining bounds on $P(\mathbf{Q})$ from bounds on $P(\mathbf{NQ})$ can also be used as the basis for a message passing algorithm in which messages are not exact, but bounded themselves, either from incomplete evaluation or from a model specification which has been bounded from the beginning.

5.4 The Anytime Bounded VE Algorithm

We devise the ABVE algorithm which calculates bounds on the query when the query neighborhood changes as a result of variable elimination. It always keeps a valid bound on the query marginal and can be interrupted

at any time. This allows us to run VE as usual or to stop as soon as the bounds reach a satisfactory width, providing an anytime algorithm that in some situations can prove much more effective than VE alone.

ABVE can use an efficient elimination ordering and update the query bounds only when a query neighborhood node is eliminated, or it can give preference to eliminating variables in the neighborhood so as to obtain sooner query bounds. The latter however may sacrifice performance since it may yield a worse elimination ordering. In fact, a tradeoff exists between ordering efficiency and sooner bounds, so methods for picking the next eliminated variable can be devised to take it into account.

We show in section 5.6 that the bounds always get tighter after the elimination of some variable in **NQ** (naturally, they are unchanged when other variables are eliminated). The algorithm stops when it is interrupted or **NQ** becomes empty (since this means that either we are left with the query alone or that query is disconnected from the rest of the model, allowing us to calculate its marginal independently from it). In the latter case the bound provided by $\Phi_{\mathbf{Q}}$ will have width zero and represent the exact solution.

5.5 Experiments

We examine the behavior of ABVE empirically by using random networks over 60 binary random variables with sparse connectivity of 2 per node and potentials limited to $[\alpha, 1-\alpha]$, for $\alpha \in \{0, 0.05, \ldots, 0.25\}$. We also use two types of 10 by 10 grid of binary random variables: conjunction grids, with potential 0.9 between neighbors both equal to 1 and 0.5 otherwise, and equality grids, or Ising model, with potential 0.9 for neighbors with equal values and 0.1 for neighbors with different values. We apply both ABVE and VE to these networks. The graph in figure 5.3 shows the tightening of the average bound interval widths vs. the time it took to reach it, as a percentage of the exact calculation time taken by VE. For the random networks, ABVE typically reaches a bound width of around 0.3 in a short time, decreasing it to about 0.2 by the time of exact calculation. Because ABVE is more restricted in its ordering choices, it takes much longer to reach the exact solution than VE. However, the experiments show that it reduces most of the uncertainty about the marginal probability in only 3% of the time taken to calculate the exact solution. This indicates ABVE not as a substitute of VE, but as a method to quickly obtain guaranteed bounds on the solution. Note that the larger α is, the tighter bounds get. This is not surprising since a larger α will create factors that locally restrict the query to a greater extent. For the grid networks, there is a stark contrast between the two types. The bounds immediately narrow to almost 0 for the conjunction grid but practically do not change in the Ising model. Again this is not surprising, because the factors in the conjunction grid have narrower potential ranges and therefore provide strong local information. In the Ising model, each variable depends

to a large extent on the specific values of its neighbors, which by their turn depend on *their* neighbors and so on. The interactions are not as local and any algorithm using local factor neighborhood alone will not be able to perform well on this model. Our implementation selects efficient elimination orderings by the greedy min-weight method [Kja93] (that is, selecting the variable whose elimination will create the factor with the least number of assignments). VE is free to select any variables but the ABVE chooses them from the query neighborhood only in order to narrow the bound as soon as possible.

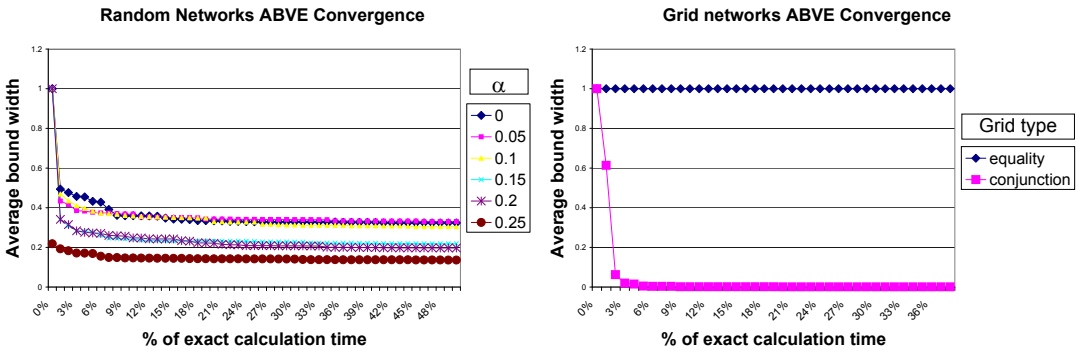

Figure 5.3: Average interval width of bounds vs. computation time over random and grid networks (see section 5.5).

5.6 Bound Updates

An important question is whether eliminating a variable in **NQ** and recalculating the bounds may result in worse bounds. We show that this is not so.

Theorem 4. Let $P^+(\mathbf{q})$ be a bound on $P(\mathbf{q})$ for a set of variables $\mathbf{Q}$ with neighboring variables $\mathbf{NQ}$ in a factor network Φ, and let $P^{+'}(\mathbf{q})$ be the bound on $P(\mathbf{q})$ in a factor network resulting from the variable elimination of $\mathbf{Y} \subseteq \mathbf{NQ}$ from Φ. Then $P^{+'}(\mathbf{q}) \leq P^+(\mathbf{q})$.

Proof. Figure 5.4 shows the groups of random variables and factors involved in the situation described. Let

- Φ_1 be the factors in $\Phi_{\mathbf{Q}}$ without any variable in $\mathbf{Y}$ as a parameter,
- Φ_2 be the factors in $\Phi_{\mathbf{Q}}$ with part of $\mathbf{Y}$ as parameters,
- Φ_3 be the factors not in $\Phi_{\mathbf{Q}}$,
- Φ_4 be all other factors,
- $\mathbf{W}_1$ be $\mathbf{NQ} \setminus \mathbf{Y}$,

- $\mathbf{W}_2$ be all variables not in $\mathbf{NQ}$ neighboring $\mathbf{Y}$,
- $\mathbf{W}_3$ be all other variables,,
- ϕ' be the factor formed by the elimination, that is, $\phi'(\mathbf{q}, \mathbf{w}_1, \mathbf{w}_2) = \sum_{\mathbf{y}} \Phi_2(\mathbf{q}, \mathbf{y}, \mathbf{w}_1)\Phi_3(\mathbf{y}, \mathbf{w}_1, \mathbf{w}_2)$.

Note that our convention of factors ignoring extra parameters is used here. A factor connected to variables in $\mathbf{Y}$ and $\mathbf{W}_2$ but not $\mathbf{W}_1$ can be included i Φ_3 because it will ignore any variables from $\mathbf{W}_1$.

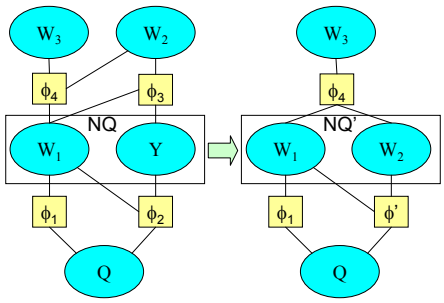

Figure 5.4: Groups of variables and factors before and after the elimination of $\mathbf{Y}$.

$\mathbf{Q}$'s neighboring variables in the new network are $\mathbf{NQ}' = \mathbf{W} = \mathbf{W}_1 \cup \mathbf{W}_2$.

According to the blanket bound

$$P^{+'}(\mathbf{q}) = \max_{\mathbf{nq}'} \frac{\phi_1(\mathbf{nq}', \mathbf{q})\phi'(\mathbf{nq}', \mathbf{q})}{\sum_{\mathbf{q}'} \phi_1(\mathbf{nq}', \mathbf{q}')\phi'(\mathbf{nq}', \mathbf{q}')}$$
$$= \max_{\mathbf{w}} \frac{\phi_1(\mathbf{w}_1, \mathbf{q})\phi'(\mathbf{w}, \mathbf{q})}{\sum_{\mathbf{q}'} \phi_1(\mathbf{w}_1, \mathbf{q}')\phi'(\mathbf{w}, \mathbf{q}')}$$

For convenience, we define $\phi_5 = \phi_1\phi_2$ and write

$$\phi_1(\mathbf{w}_1, \mathbf{q})\phi'(\mathbf{w}_1, \mathbf{q}) = \phi_1(\mathbf{w}_1, \mathbf{q}) \sum_{\mathbf{y}} \phi_2(\mathbf{w}_1, \mathbf{y}, \mathbf{q})\phi_3(\mathbf{y}, \mathbf{w}_2)$$
$$= \sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})\phi_3(\mathbf{y}, \mathbf{w}_2),$$

and

$$P^{+'}(\mathbf{q}) = \max_{\mathbf{w}} \frac{\sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})\phi_3(\mathbf{y}, \mathbf{w}_2)}{\sum_{\mathbf{q}'} \sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q}')\phi_3(\mathbf{y}, \mathbf{w}_2)}$$

$$= \max_{\mathbf{w}_1} \frac{\sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})\phi_6(\mathbf{y})}{\sum_{\mathbf{q}'} \sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q}')\phi_6(\mathbf{y})} \text{ (by picking max } \mathbf{w}_2)$$

By dividing both numerator and denominator by the constant $\sum_{\mathbf{y}'} \phi_6(\mathbf{y}')$, we obtain a probability distribution $\mathbf{p}$ on $\mathbf{Y}$.

$$= \max_{\mathbf{w}_1} \frac{\sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})\mathbf{p}(\mathbf{y})}{\sum_{\mathbf{q}'} \sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q}')\mathbf{p}(\mathbf{y})}$$

Because $\mathbf{p}$ is a distribution, $\sum_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})\mathbf{p}(\mathbf{y}) \leq \max_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})$. And because the quotient of linear functions are monotonic,

$$\leq \max_{\mathbf{w}_1} \frac{\max_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})}{\sum_{\mathbf{q}'} \max_{\mathbf{y}} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q}')} = \max_{\mathbf{w}_1, \mathbf{y}} \frac{\phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q})}{\sum_{\mathbf{q}'} \phi_5(\mathbf{w}_1, \mathbf{y}, \mathbf{q}')}$$

$$= \max_{\mathbf{nq}} \frac{\phi_1(\mathbf{nq}, \mathbf{q})\phi_2(\mathbf{nq}, \mathbf{q})}{\sum_{\mathbf{q}'} \phi_1(\mathbf{nq}, \mathbf{q}')\phi_2(\mathbf{nq}, \mathbf{q}')} = P^+(\mathbf{q}) \qquad \square$$

5.7 Related Work

Approximate inference in graphical models is NP-hard [DL93], but many algorithms have been developed for it nonetheless using many different techniques. Variational models [JGJS99] approximate a distribution by assuming it is similar in behavior to a simpler one. Sampling methods [DL97] simulate the model and estimate probabilities from the observed sequences. Search-based methods [Poo96] gradually evaluate parts of the assignment space of a model, seeking the most relevant ones and often deriving bounds from them. Some methods rely on ignoring dependencies when factors grow too large [DR03, Lar03]. [LW96] goes from coarser to finer value space granularity in the model in order to derive increasingly better bounds. [Poo98] uses the inner structure of factors to simplify them and ignores finer distinctions in order to make computation cheaper. [HSC89] uses Pearl's [Pea88] cutset conditioning method in order to split a model into polytrees (which require linear time computation only) and then evaluate the most likely ones first in order to derive bounds. The aforementioned [LK03] uses linear programming on constraints posed by neighboring regions of the query in order to derive bounds (but may not converge to the exact solution). [BD06] combines both [HSC89] and [LK03].

Inspired by [Nil86], an entire line of research seeks to perform probabilistic inference via inference rules not unlike logical ones [FHM90, FH94] and often produce partial, bounded results.

Some works are based on the assumption that the initial probabilities will be specified within an interval only or via constraints (credal sets) [Luk99, FZ98]. Therefore, it is natural for them to deal with bounds that get tighter with further processing. However, they differ from our work in the sense that they are based on an model intrinsically involving bounds to start with.

Several of the methods above [DR03, Lar03, HSC89, BD06, Poo96, Poo98] are anytime bounded approximation algorithms like ABVE, but the ideas involved differ significantly. Due to this, ABVE can be seen as an orthogonal and complementary rather than alternative method to them. Because of its conceptual and implementational simplicity, it can be integrated and compounded with them, bringing the benefits of its own approximation technique, and dependence on the query neighborhood only.

5.8 Future Directions

Many future directions regarding the blanket bound and ABVE remain to be taken. An important theoretical contribution would involve the use of some measure of maximum skewness of factors, or a measure of their determinism, to provide an analytical prediction of bound tightening as a function of distance from the query. Also, the algorithm currently picks an arbitrary neighbor to be eliminated, but we believe heuristics can be designed for selecting them so that bounds are more efficiently tightened.

We intend to apply ABVE to applications in which graphical models are dynamically built from a relational, or first-order, specification, making the link to logical inference even closer. Another step in this direction is to investigate the possibility of eliminating a variable from only a *pair* of factors at a time, as done in logical resolution, as opposed to doing it from *all* factors having that variable as a parameter.

5.9 Conclusion

We presented an optimal marginal probability bound based on the neighboring factors of a query, and an anytime bounded approximation probabilistic inference algorithm that, in a way analogous to logical inference, takes advantage of the fact that certain parts of a graphical model are more relevant and can greatly bound inference results. While useful as an approximation method in itself, ABVE is distinguished from other approximated probabilistic inference methods by requiring knowledge of the query neighborhood only in order to derive bounds on its marginal probability. This is also analogous to logical inference that only considers clauses immediately related to the query. This is particularly useful in large graphical models which are dynamically built or stored away, since they can be gradually processed until a satisfactory bound is found. Besides its practical benefits, ABVE contributes to the better understanding of the relation between logical and probabilistic inferences.

Chapter 6

Conclusion

Intuitive descriptions of models very often include first-order elements. When these models are probabilistic, the dominant approach has been that of grounding the model to a propositional one and solving it with a regular propositional algorithm. This strategy loses the explicit representation of the model's first-order structure, which can be used to great computational advantage, and which is computationally hard to retrieve from the grounded model.

We have presented FOVE, a first-order generalization of the popular Variable Elimination propositional inference algorithm. Like VE, FOVE successively eliminates random variables from the model by summing them out while taking advantage of independences for efficiency. Unlike VE, FOVE directly manipulates first-order representations, eliminating c-atoms that stand for potentially large sets of random variables at once. This can in some cases exponentially (in the domain size) speed inference up.

We have also presented the necessary modifications for FOVE to solve the MPE problem, another important problem in probabilistic inference.

FOVE requires a preprocessing step (shattering, presented in Chapter 3) over the entire model before inference. In certain situations this may render inference more expensive than it needs to be. We have presented the first step towards a more gradual processing of the model with the Anytime Bounded Variable Elimination method in Chapter 5. The method is currently propositional but has been develop with the purpose of future expansion to first-order.

6.1 Future Directions

There are several possible directions for further development of FOVE. One of the main ones is the incorporation of function symbols, both uninterpreted and therefore subject to the probability distribution (the color of objects, for example) and interpreted (summation over integers), which will greatly increase its expressivity and applicability.

In applications involving evidence over many objects (for example, the facts about all the words in an

English document), shattering may take a long time because all parfactors have to be checked against it. The large number of objects involved may create the need for numerous parfactor splittings. This is unfortunate because often only some objects are truly relevant to the query. For example, analyzing only some words and phrases in a document will often be enough to determine its subject. Therefore a variant of FOVE that does only the necessary shattering, guided by the inference process, is of great interest.

It is also interesting to think of FOVE from a logical programming point of view. In logic programming, the result of an inference is the binding of logical variables to values satisfying a goal. This could be brought to FOVE by allowing queries that are not ground (for example, $sick(P)$), and obtaining the query marginals as a function of logical variable bindings (for example, $P(sick(john)) = 0.3, P(sick(mary)) = 0.4$ etc). We could also simply ask for the most likely binding of logical variables in a query.

Currently, FOVE operates on domains with a fixed, known number of objects. An interesting research direction is that of adopting an open world assumption, as BLOG ([MMR+05]) does.

Finally, lifted FOVE operations do not cover all possible cases and explicit summation may be required at times, so increasing their coverage is an important direction.

In spite of its current limitations, FOVE is already applicable, and especially useful in domains with a large number of objects about which we have identical knowledge. More importantly than that, it is a general framework to be expanded and help close the gap between logic and probabilistic reasoning.

Appendix A

Uniform Solution Counting Partition

In operations such as partial inversion and alignment of parfactors, we may need to calculate $|[C'\theta]|$ for each $\theta \in [C]$. for C, C' two constraint systems.

For example, let C be $X \neq a$ and C' be $Y \neq a \wedge Y \neq b$. We may need to calculate the number of solutions of $(Y \neq a \wedge Y \neq b)\theta$ for each $\theta \in [C]$. This is a trivial case in which the answer is $|Y| - 1$ regardless of θ.

However, more complicated cases arise when variables in C' depend on variables in C. For $C \equiv X \neq a$ and $C' \equiv Y \neq a, Y \neq X$, $|[C'\theta]|$ does depend on the particular θ: it is $|Y| - 1$ when θ assigns a to X and $|Y| - 2$ otherwise. We are therefore motivated to find a set of constraint systems $C_1, \ldots, C_n$ such that $[C_1], \ldots, [C_n]$ is a partition of $[C]$, such that every $\theta \in C_i$ induces the same solution size of $|[C'\theta]|$.

This motivates the following definition:

Definition 1 (Uniform solution counting partition)**.** Let C, C' be two constraint systems. The *uniform solution counting partition of C with respect to C'* is a set of constraint systems $\{C_1, \ldots, C_k\}$ where $\{[C_1], \ldots, [C_k]\}$ is a partition of $[C]$ and

$$\forall i \in \{1, \ldots, k\} \cdot \forall \theta', \theta'' \in [C_i] \cdot |[C'\theta']| = |[C'\theta'']|.$$

In fact, it is also necessary to define the same concept with respect to a *set* of constraint systems:

Definition 2 (Uniform solution counting partition for a set of constraint systems)**.** Let C be a constraint system and $\mathbf{C}$ a set of constraint systems. The *uniform solution counting partition of C wrt $\mathbf{C}$* is a set of constraint systems $\{C_1, \ldots, C_k\}$ where $\{[C_1], \ldots, [C_k]\}$ is a partition of $[C]$ and

$$\forall i \in \{1, \ldots, k\} \forall C' \in \mathbf{C} \cdot \forall \theta', \theta'' \in [C_i] \cdot |[C'\theta']| = |[C'\theta'']|.$$

The specifics of obtaining these partitions depends on the constraint language used.

Appendix B

Fusion

We now explain how a set of parfactors G can be replaced by a single, equivalent parfactor.

Definition 3. Let G be a set of parfactors. The *fusion of* G, $fs(G)$, is defined as the parfactor (ϕ', A_G, C_G), with $\phi'(A_G\theta) = \prod_{g \in G} \phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|}$ for any $\theta \in \Theta_G$.

Theorem 5. For any set of parfactors G, $\Phi(G) = \Phi(fs(G))$.

Proof.

$$\begin{aligned}
\Phi(G) &= \prod_{g \in G} \prod_{\theta \in \Theta_g} \phi_g(A_g\theta) = \prod_{g \in G} \prod_{\theta \in \Theta_G} \phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|} \\
&= \prod_{\theta \in \Theta_G} \prod_{g \in G} \phi_g(A_g\theta)^{|\Theta_g|/|\Theta_G|} \\
&= \prod_{\theta \in \Theta_G} \phi'(A_G\theta) = \Phi(fs(G))
\end{aligned}$$

□

The crucial step is the one in which we replace each original set of constraint solutions Θ_g by the global constraint solution set Θ_G. When this happens, each original instantiation of a parfactor is now instantiated $|\Theta_G|/|\Theta_g|$ many times more than before, but the power $|\Theta_g|/|\Theta_G|$ preserves the original potential value.

B.0.1 Parfactor Alignment

The fusion method above, while correct, may sometimes create fused parfactors which are unnecessarily complex, as the following example shows.

Example B.0.1. Given parfactors $g_1 = (\phi_1, \{p(X, W)\}, X \neq v_1)$ and $g_2 = (\phi_2, q(Y, Z), Y \neq v_1, Z \neq v_n)$, with X, W, Y, Z with domain $D = \{v_1, \ldots, v_n\}$, the fused parfactor is on atoms $p(X, W), q(Y, Z)$, with a number of instantiations equal to roughly $|D|^4$.

However, through a technique we call *alignment*, one can obtain an equivalent but simpler parfactor. Alignment exploits the fact that, sometimes, a subset of logical variables in the parfactors being fused are

defined on the same domain and can be unified, decreasing the number of logical variables in the fused parfactor. In this example, we can align the variables X and Y:

$$
\begin{aligned}
&\big(\prod_{X\neq v_1,W}\phi_1(p(X,W))\big)\big(\prod_{Y\neq v_1,Z\neq v_n}\phi_2(q(Y,Z))\big)\\
=&\big(\prod_{W}\phi_1(p(v_2,W))\big)\ldots\big(\prod_{W}\phi_1(p(v_n,W))\big)\big(\prod_{Z\neq v_n}\phi_2(q(v_2,Z))\big)\ldots\big(\prod_{Z\neq v_n}\phi_2(q(v_n,Z))\big)\\
=&\big(\prod_{W}\phi_1(p(v_2,W))\big)\big(\prod_{Z\neq v_n}\phi_2(q(v_2,Z))\big)\ldots\big(\prod_{W}\phi_1(p(v_n,W))\big)\big(\prod_{Z\neq v_n}\phi_2(q(v_n,Z))\big)\\
=&\prod_{X\neq v_1}\big(\prod_{W}\phi_1(p(X,W))\big)\big(\prod_{Z\neq v_n}\phi_2(q(X,Z))\big)\\
=&\prod_{X\neq v_1}\prod_{W,Z\neq v_n}\phi_1(p(X,W))^{|[W]|/|[W,Z\neq v_n]|}\phi_2(q(X,Z))^{|[Z\neq v_n]|/|[W,Z\neq v_n]|}\\
=&\prod_{X\neq v_1}\prod_{W,Z\neq v_n}\phi_3(p(X,W),q(X,Z))\\
=&\prod_{X\neq v_1,W,Z\neq v_n}\phi_3(p(X,W),q(X,Z))
\end{aligned}
$$

which is the potential of a parfactor with about $|D|^3$ instantiations. X and Y have been *aligned*.

Note that it is important that the number of solutions of $W \wedge Z \neq v_n$ does not depend on the current value of X, allowing us to define a function ϕ_3 that is the same under any X. This condition will be enforced in the formalization below.

Definition 4 (Fusion with alignment)**.** Let G be a set of parfactors.

A *fsa tuple* of G is a tuple $(\mathbf{X}, \theta_{.\to\mathbf{X}})$ where

- $\mathbf{X}$ is a set of logical variables (possibly empty), called *the normal set*,
- $\theta_{.\to\mathbf{X}}$ is a function from each $g \in G$ to a one-to-one substitution $\theta_{g\to\mathbf{X}}$

based on which we define, for convenience,

- $\mathbf{X}_g = Dom(\theta_{g\to\mathbf{X}})$
- $C'_g \equiv C_g\theta_{g\to\mathbf{X}}$ (the *normalized constraint of* g)

such that

1. there is a constraint system $C_{\mathbf{X}}$ such that $C_{g|\mathbf{X}_g}\theta_{g\to\mathbf{X}} \Leftrightarrow C_{\mathbf{X}}$ for all $g \in G$;

2. the uniform solution counting partition (appendix A) of $C_{\mathbf{X}}$ wrt C'_g is $\{C_{\mathbf{X}}\}$, for each $g \in G$. That is,

$$\forall g \in G \forall \theta', \theta'' \in C_{\mathbf{X}} \cdot |[C'_g\theta']| = |[C'_g\theta'']|;$$

3. the uniform solution counting partition of $C_{\mathbf{X}}$ wrt $\bigwedge_{g\in G} C'_g\theta_{\mathbf{X}}$ is $\{C_{\mathbf{X}}\}$, where $\theta_{\mathbf{X}}$ is any element of $[C_{\mathbf{X}}]$.

If G has a fsa tuple $(\mathbf{X}, \theta_{.\to\mathbf{X}})$, then we define $fsa(G, \mathbf{X}, \theta_{.\to\mathbf{X}})$, the *fusion with alignment of G wrt* $(\mathbf{X}, \theta_{.\to\mathbf{X}})$, as the parfactor (ϕ', A', C') where

$$\begin{aligned} A' &= \bigcup_{g\in G} A_g\theta_{g\to\mathbf{X}} \\ \phi'(A') &= \prod_{g\in G} \phi_g(A_g)^{|[C'_g\theta_{\mathbf{X}}]|/|[\bigwedge_{g'\in G} C'_{g'}\theta_{\mathbf{X}}]|} \\ C' &= C_{\mathbf{X}} \wedge \bigwedge_{g\in G} C'_g\theta_{\mathbf{X}} \end{aligned}$$

where $\theta_{\mathbf{X}}$ is any element of $[C_{\mathbf{X}}]$.

In Example B.0.1, the fsa tuple used was

- $\mathbf{X} = \{X\}$
- $\theta_{1\to\mathbf{X}} = \{(X, X)\}$
- $\theta_{2\to\mathbf{X}} = \{(Y, X)\}$

with $C_{\mathbf{X}} \equiv X \neq v_1$.

Theorem 6. Let G be a set of parfactors. If G has a fsa tuple $(\mathbf{X}, \theta_{.\to\mathbf{X}})$, then $\Phi(G) = \Phi(fsa(G, \mathbf{X}, \theta_{.\to\mathbf{X}}))$.

Proof. Let A', ϕ', C' be as in Definition 4.

$$\begin{aligned} &\Phi(G) \\ &= \prod_{g\in G} \prod_{\theta\in[C_g]} \phi_g(A_g\theta) \\ &= \prod_{g\in G} \prod_{\theta_{\mathbf{X}_g}\in[C_{g|\mathbf{X}_g}]} \prod_{\theta\in[C_g\theta_{\mathbf{X}_g}]} \phi_g(A_g\theta_{\mathbf{X}_g}\theta) \end{aligned}$$

(by normalizing all $\mathbf{X}_g$ to $\mathbf{X}$)

$$= \prod_{g \in G} \prod_{\theta_{\mathbf{X}} \in [C_{\mathbf{X}}]} \prod_{\theta \in [C_g \theta_{g \to \mathbf{X}} \theta_{\mathbf{X}}]} \phi_g(A_g \theta_{g \to \mathbf{X}} \theta_{\mathbf{X}} \theta)$$
$$= \prod_{g \in G} \prod_{\theta_{\mathbf{X}} \in [C_{\mathbf{X}}]} \prod_{\theta \in [C'_g \theta_{\mathbf{X}}]} \phi_g(A' \theta_{\mathbf{X}} \theta)$$

(since $[C_{\mathbf{X}}]$ does not depend on g)

$$= \prod_{\theta_{\mathbf{X}} \in [C_{\mathbf{X}}]} \prod_{g \in G} \prod_{\theta \in [C'_g \theta_{\mathbf{X}}]} \phi_g(A' \theta_{\mathbf{X}} \theta)$$

(by replacing each $[C'_g \theta_{\mathbf{X}}]$ by the conjunction of all of them, and because of conditions 2 and 3 in Definition 4)

$$= \prod_{\theta_{\mathbf{X}} \in [C_{\mathbf{X}}]} \prod_{g \in G} \prod_{\theta \in [\bigwedge_{g' \in G} C'_{g'} \theta_{\mathbf{X}}]} \phi_g(A' \theta_{\mathbf{X}} \theta)^{|[C'_g \theta_{\mathbf{X}}]|/|[\bigwedge_{g' \in G} C'_{g'} \theta_{\mathbf{X}}]|}$$

(since that conjunction does not depend on the particular g)

$$= \prod_{\theta_{\mathbf{X}} \in [C_{\mathbf{X}}]} \prod_{\theta \in [\bigwedge_{g' \in G} C'_{g'} \theta_{\mathbf{X}}]} \prod_{g \in G} \phi_g(A' \theta_{\mathbf{X}} \theta)^{|[C'_g \theta_{\mathbf{X}}]|/|[\bigwedge_{g' \in G} C'_{g'} \theta_{\mathbf{X}}]|}$$
$$= \prod_{\theta_{\mathbf{X}} \in [C_{\mathbf{X}}]} \prod_{\theta \in [\bigwedge_{g' \in G} C'_{g'} \theta_{\mathbf{X}}]} \phi'(A' \theta_{\mathbf{X}} \theta)$$
$$= \prod_{\theta \in [C_{\mathbf{X}} \wedge \bigwedge_{g' \in G} C'_{g'} \theta_{\mathbf{X}}]} \phi'(A' \theta)$$
$$= \prod_{\theta \in [C']} \phi'(A' \theta)$$
$$= \Phi(fsa(G, \mathbf{X}, \theta_{\cdot \to \mathbf{X}}))$$

□

Appendix C

Shattering

The elimination of atoms requires certain conditions guaranteed by the fact that the set of parfactors having been *shattered* against the query. Shattering is an extension of notions discussed in [Poo03].

Definition 5 (Shattering)**.** A set of parfactors is *shattered* if, for every pair of atoms (p, q) in G, their groundings $RV(p)$ and $RV(q)$ are either identical or disjoint.

Example C.0.2. Parfactors $(\phi_1, p(X, a), \top)$ and $(\phi_2, p(b, Y), Y \neq d)$ are not shattered because $RV(p(X, a))$ and $RV(p(b, Y))$ overlap but are not identical, violating (1).

The algorithm in figure C.1 shatters a set of parfactors against a query. It works by repeatedly identifying pairs of improper pairs and *breaking* parfactors into equivalent sets of parfactors whose sets of instantiations are the same as the original ones, but inducing proper pairs. This is done by, through unification, determining the conditions for the groundings of improper pairs to coincide or not, and breaking the parfactors along these conditions. After this, unified atoms are normalized (see Appendix D).

Example C.0.3. If we have parfactor $(\phi_2, p(b, Y), Y \neq d)$ and query $p(b, c)$, $p(b, Y)$ and $p(b, c)$ are an improper pair. Their most general unifier (MGU) is $Y = c$, so we can break the parfactor into $(\phi_2, p(b, Y), Y \neq d \wedge Y = c)$ and $(\phi_2, p(b, Y), Y \neq d \wedge Y \neq c)$ which can be normalized as $(\phi_2, p(b, Y), Y = c) = (\phi_2, p(b, c), \top)$ and $(\phi_2, p(b, Y), Y \neq d \wedge Y \neq c)$.

FUNCTION *SHATTER*(G,Q)
G a set of parfactors, Q a set of atoms.

1. If there exists an improper atom pair p, q in $A_G \cup Q$
   (a) For each $r \in \{p, q\}$
   If r comes from parfactor g
   i. $g' \leftarrow NORMALIZE(\phi_g, A_g, C_g \wedge MGU(p, q))$.
   ii. $g'' \leftarrow NORMALIZE(\phi_g, A_g, C_g \wedge \neg MGU(p, q))$.
   iii. $G \leftarrow (G \setminus \{g\}) \cup \{g', g''\}$.
   (b) Return $SHATTER(G, Q)$.
2. Return G.

FUNCTION *NORMALIZE*(g)
g a parfactor.

1. If there exists a pair of atoms p, q in A_g such that $C_g \Rightarrow p = q$
   replace q by p in g.
2. Return $NORMALIZE(g)$.

Figure C.1: Shattering algorithm.

Appendix D

Parfactor Normalization

At several points it is convenient to have parfactors represented in a normalized form that does not involve constraint systems with logical variables which are not present in the atoms, or with unnecessary equalities.

Example D.0.4. Below are two parfactors and their normalized versions:

Parfactor	Normalized parfactor
$(\phi, \{p(X), q(Y)\}, X \neq Y, Z \neq a)$	$(\phi^{\lvert[Z\neq a]\rvert}, \{p(X), q(Y)\}, X \neq Y)$
$(\phi, \{p(X), q(Y)\}, X = Y)$	$(\phi, \{p(X), q(X)\}, \top)$

Definition 6 (Cleaned-up parfactor)**.** Let g be a parfactor. The *cleaned-up version of* g is denoted $cl(g)$. If the uniform solution counting partition of $C_{g_{\mid LV(A_g)}}$ wrt C_g is $\{C_{g_{\mid LV(A_g)}}\}$ (section A), $cl(g)$ is defined as the parfactor $(\phi_g^{\lvert[C_g\theta]\rvert}, A_g, C_{\mid LV(A_g)})$ where θ is an arbitrary element of $[C_{\mid LV(A_g)}]$. Otherwise, $cl(g)$ is defined as g itself.

Theorem 7. Let g be a parfactor. Then $\Phi(g) = \Phi(cl(g))$.

Proof.

$$
\begin{aligned}
\Phi(g) &= \prod_{\theta \in [C_g]} \phi(A_g\theta) \\
&= \prod_{\theta \in [C_{g\mid LV(A)}]} \prod_{\theta' \in [C_g\theta]} \phi(A_g\theta\theta') \\
&= \prod_{\theta \in [C_{g\mid LV(A)}]} \prod_{\theta' \in [C_g\theta]} \phi(A_g\theta) \\
&= \prod_{\theta \in [C_{g\mid LV(A)}]} \phi^{\lvert[C_g\theta]\rvert}(A_g\theta) \\
&= \prod_{\theta \in [C_{g\mid LV(A)}]} \phi_{cl(g)}(A_g\theta) \\
&= \Phi(cl(g))
\end{aligned}
$$

□

Definition 7 (Equalizing substitution and equalization)**.** Let O be some total ordering of terms in which constant terms are always greater than any logical variable. Let C be a constraint system. For each variable $X \in LV(C)$, let $root(X)$ be the term t such that t is the maximal term (according to O) satisfying $X \leq t$ and $(\varphi(C) \Rightarrow X = t)$. The *equalizing substitution of* C, denoted $\theta_{=,C}$, is the composition of all substitutions of each logical variable $X \in LV(C)$ by $root(X)$.

Let g be a parfactor such that $LV(C_g) = LV(A_g)$ (that is, it is a cleaned-up parfactor). The *equalization of* g, denoted $g_=$, is the parfactor $(\phi_g, A', C_{g|LV(A')})$, where $A' = A_g\theta_{=,C_g}$.

Theorem 8. Let g be a parfactor. Then $\Phi(g) = \Phi(g_=)$.

Proof.

$$\Phi(g) = \prod_{\theta \in [C_g]} \phi_g(A_g\theta)$$

(because each replacement $(X, root(X))$ in $\theta_{=,C}$ is such that $\varphi(C) \Rightarrow X = root(X)$, θ must map both X and $root(X)$ to the same value, so replacing A_g by A' does not change anything)

$$= \prod_{\theta \in [C_g]} \phi_g(A'\theta)$$

(by projecting onto $LV(A')$)

$$= \prod_{\theta' \in [C_{g|LV(A')}]} \prod_{\theta'' \in [C_g\theta']} \phi_g(A'\theta'\theta'')$$

(because $LV(A')$ does not contain logical variables in the domain of θ'')

$$= \prod_{\theta' \in [C_{g|LV(A')}]} \prod_{\theta'' \in [C_g\theta']} \phi_g(A'\theta')$$

(because the identity of θ'' is now irrelevant)

$$= \prod_{\theta' \in [C_{g|LV(A')}]} \phi_g(A'\theta')^{|[C_g\theta']|}$$

(because $LV(C_g) = LV(A)$, all logical variables in $LV(C_g) \setminus LV(A')$ must have a root in $LV(A')$ and are therefore constrained once θ' has been chosen, so $|[C_g\theta']| = 1$)

$$= \prod_{\theta' \in [C_{g|LV(A')}]} \phi_g(A'\theta')$$
$$= \Phi(g_=)$$

□

Example D.0.5 (Equalization)**.** Let g be the parfactor

$$(\phi, \{p(X), q(Y), r(Z), s(U), t(V)\}, X = Y \wedge Y = Z \wedge Z = a \wedge U = V \wedge U \neq b).$$

Using an alphabetical ordering of logical variables,

$$g_= = (\phi, \{p(a), q(a), r(a), s(V), t(V)\}, V \neq b).$$

Definition 8 (Normalization)**.** Let g be a parfactor. The *normalization of* g, denoted $normal(g)$, is defined as $cl(g)_=$.

Theorem 9. Let g be a parfactor. Then $\Phi(g) = \Phi(normal(g))$.

Proof. This proof follows immediately from Theorems 7 and 8. The condition posed by the latter that $LV(C_{cl(g)}) = LV(A_{cl(g)})$ is satisfied since $C_{cl(g)} = C_{g|LV(A_g)}$. □

Appendix E

Complexity Analysis

In this chapter, we develop a complexity analysis of FOVE. We do this by first calculating the complexity of each operation, and combining them into a general FOVE complexity.

This analysis is a relatively superficial one that must be taken as a potentially very large overestimate, since it does not take into account possible amortizations between many operations. Better analyses remain a future research direction.

E.1 Individual Operation Complexities

Let us analyze the complexity of individual operations summing out a set of c-atoms E from a parfactor g, with $a = |A_g|$, $e = |E|$, $r = \max_{A \in A_g} |RV(A)|$, c the size of C_g and $d = \max_{A \in A_g} |D_A|$.

E.1.1 Propositionalization

Because it not always possible to use lifted inference operations in FOVE, sometimes we have to resort to propositionalization of segments of a model. In these cases, the cost is exponential in the tree width of the segment, and can be as high as exponential in the number of ground random variables in it.

E.1.2 Counting Elimination

Counting Elimination starts by verification of atom independence. This requires the solving of satisfiability of a constraint for each pair of atoms in E, and therefore takes time $O(e^2 2^c)$.

Counting elimination also requires the calculation of $|RV(A)|$ for each atom A in E. Each of these calculations can be done in time $O(2^c)$, so this takes $O(e2^c)$ for all atoms in E.

Counting elimination proper iterates over $\vec{N}_E$, the multinomial counter of atoms in E. These counters correspond to lifted assignment to these atoms (see Chapter 4). Therefore, there will be as many multinomial counters for an atom A as there are lifted assignments on it. A lifted assignment on A can be seen as a multiset of cardinality $RV(A)$ drawn from a set of D_A possible elements. The number of such multisets is the

multiset coefficient $\left\langle {|D_A| \atop |RV(A)|} \right\rangle$, equal to $\binom{|D_A|+|RV(A)|-1}{|D_A|}$, which is $O((|D_A|+|RV(A)|-1)^{\min(|D_A|,|RV(A)|-1)})$. Therefore nested iteration over all of them is $O\Big(\big((d+r-1)^{\min(d,r-1)}\big)^a\Big)$.

In each multinomial iteration, one needs to compute the multinomial coefficient $\vec{N}_E!$ and calculate the product of $\phi_g(v)^{\#(v,\vec{N}_E)}$ for all assignments v to A_g. While the calculation of an individual multinomial coefficient is expensive, we can obtain the current coefficient from the previous one by a constant-time incremental update. The calculation of $\phi_g(v)$ and $\#(v,\vec{N}_E)$ are lookups of the potential table of ϕ_g and the current multinomial $\vec{N}_E$, indexed by the current assignment v. Again this can be done in constant time per iteration of v by keeping a pointer to the current position of each of those tables, and updating the pointer at each iteration of v. So this part of the cost is simply the number of values of v, $O(d^a)$.

Therefore, the complexity of a single counting elimination operation is

$$O\Big(e2^c + e^2 2^c + \big((d+r-1)^{\min(d,r-1)}\big)^a d^a\Big).$$

E.1.3 Inversion

The complexity of Inversion is best done while analyzing two distinct cases. If we invert on all logical variables in g, the operation starts by a straightforward propositional summation, since all logical variables will be bound and all atoms will be propositional. This, like propositional VE, requires time d^a. The check for whether we can invert on all logical variables is cheap, consisting in identifying an atom containing all logical variables in the parfactor. After that we eliminate spurious logical variables from the constraint system and calculate the resulting reduction in the number of groundings, which involves solving a constraint system, and takes time $O(2^c)$.

If on the other hand a partial Inversion is performed, we have a sequence of m distinct inversions that result in a Counting Elimination problem (if it resulted in a propositional summation, we could have combined all inversions into a single complete one to start with). We now proceed in analyzing this case.

The first step consists in selecting the set of logical variables on which to base it, and performing the inversion proper by binding these logical variables and calculating the resulting summation as a FOVE subproblem.

While there are good heuristics for selecting the set of logical variables to be inverted (for example, logical variables shared among atoms must be inverted), in general this selection involves $2^{|LV(g)|}$, which is bounded by 2^c. For each of these sets, a check of condition 3.2 is necessary, which involves solving a constraint system for each pair of atoms in the eliminated atoms E. Therefore the selection of logical variables costs $O(e^2 2^{2c})$.

Once a set of logical variables satisfies condition 3.2, Inversion is performed by binding those variables and

repeating the process until Counting Elimination is possible. At this point one needs to calculate the size of groundings for each atom with bounded logical variables, which will cost $O(e2^c)$. The number of Inversions m is bounded by $|LV(g)|$ and therefore by c, so the total cost of this type of sequence of Inversions is $O(ce2^c)$.

Therefore, a sequence of partial inversions with counting elimination at the end costs

$$O\Big(e^2 2^{2c} + c\, e2^c + e^2 2^c + \big((d + r' - 1)^{\min(d,r'-1)}\big)^a d^a\Big)$$

where r' is the maximum grounding set size after the inverted variables are bound.

E.1.4 Shattering

Before Inversion and Counting Elimination can be applied to a set of parfactors, we need to make sure they are shattered, that is, their atoms' groundings either do not intersect or are identical.

While the algorithm in appendix C is the simplest to understand and implement, it is better to consider a modification in order to analyze its complexity. Instead of actually splitting parfactors as shattering is performed, we can instead incrementally compare pairs of c-atoms A_1 and A_2 with respective constraints C_1 and C_2 and solve constraints $C_1 \wedge C_2$, $C_1 \wedge \neg C_2$ and $\neg C_1 \wedge C_2$. These constraints determine the constraints of c-atoms deriving from A_1 and A_2 if their respective parfactors were to be split against each other. Once we do that, we can bring in a third atom A_3 associated to its own region C_3, and compare this region to the three already determined ones. This could produce a maximum of total 9 regions. It is easy to see that the maximum total number of splittings is $3 + \cdots + 3^{n-1} = \frac{3(3^{n-1}-1)}{2}$, where n is the total number of atoms in the entire model.

Once this process is completed, and assuming we keep track of all regions into a c-atom's original atom has been split to, we can then split the original parfactors into a set of shattered parfactors.

As an implementation side note, we point out that many of these splittings would actually be unnecessary. For example, if a new region R is identical to a previous region S, we already know it will not intersect with other regions S' that originated from the same splitting as S, since they will not intersect at all. Taking this into account makes the analysis more complex and we leave it to future research.

Each splitting requires solving a constraint, which has cost dependent on the size of this constraint. Because successive splittings can make constraints larger than any original constraint, we need to estimate what maximum size a constraint can be during this process. A safe bound to this size is the size of the conjunction formed from the set of of all original constraints and their negations, since every constraint derived in the process will necessarily be equivalent to a conjunction of a subset of them (although typically

it will be a simplified and smaller form). Let k be this size. Then the cost of shattering is $O(\frac{3(3^{n-1}-1)}{2}2^k)$

E.2 The FOVE Algorithm Complexity

Before analyzing the complexity of the FOVE algorithm, it is worthwhile recalling how the complexity of the propositional Variable Elimination algorithm is determined. Each variable elimination step has a cost exponential in the number of random variables it involves (its *clique size*). Given an elimination ordering, the cost of the algorithm is exponential in the largest clique size during elimination, and this clique size is called the *induced tree width* of the model. We define the *tree width* of a model as its minimum induced tree width.

For FOVE, the same reasoning takes place; different operation orderings will yield different costs. However in this case the costs are very different functions depending on the type of operation. One can then define a parameter analogous to induced tree width expressing the maximum cost given an ordering, and another, analogous to tree width, expressing the cost of the best ordering.

Differently from the tree width measure, however, is the fact that our measures are overestimates, whereas a propositional model cannot be solved exactly in time less than exponential on its tree width. Finding narrower bounds for FOVE's cost is a much more complex problem to be tackled in future research.

While determining the smallest tree width of graph is an intractable problem, one can run the propositional VE algorithm without numerical computations in order to select a good ordering. This method is extendable to FOVE – one can run it without the actual iterations for determining numerical values in order to determine the cost of a given ordering, and thus choose a good one.

References

[ADW02] Corin R. Anderson, Pedro Domingos, and Daniel S. Weld. Relational Markov models and their application to adaptive web navigation. In *KDD '02: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 143–152, New York, NY, USA, 2002. ACM Press.

[Ang02] Nicos Angelopoulos. Probabilistic finite domains: A brief overview. In *ICLP '02: Proceedings of the 18th International Conference on Logic Programming*, page 475, London, UK, 2002. Springer-Verlag.

[Bac90] Fahiem Bacchus. *Representing and reasoning with probabilistic knowledge: a logical approach to probabilities*. MIT Press, Cambridge, MA, USA, 1990.

[Bak79] J.K. Baker. Trainable grammars for speech recognition. In *Speech communication papers presented at the 97th Meeting of the Acoustical Society*, pages 547–550, 1979.

[Bau72] L. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of markov processes. *Inequalities*, 3:1–8, 1972.

[BD06] Bozhena Bidyuk and Rina Dechter. An anytime scheme for bounding posterior beliefs. In *Proceedings of the 21sth National Conference on Artificial Intelligence (AAAI)*, 2006.

[Bes75] J. Besag. Statistical analysis of non-lattice data. *The Statistician*, 24:3:179–195, 1975.

[BGR04] Chitta Baral, Michael Gelfond, and J. Nelson Rushton. Probabilistic reasoning with answer sets. In *LPNMR*, pages 21–33, 2004.

[Bre91] John S. Breese. Construction of belief and decision networks. *Computational Intelligence*, 8:624–647, 1991.

[BS84] B. Buchanan and E. Shortliffe. *Rule-Based Expert Systems: The MYCIN experiments of the Stanford Heuristic Programming Project*. Addison-Wesley, Reading, MA, 1984.

[Bun94] Wray L. Buntine. Operations for learning with graphical models. *Journal of Artificial Intelligence Research*, 2:159–225, 1994.

[Car50] R. Carnap. *The Logical Foundations of Probability*. University of Chicago Press, Chicago, 1950.

[CM82] K. L. Clark and F. G. McCabe. Prolog: A language for implementing expert systems. In J. E. Hayes, D. Michie, and Y.-H. Pao, editors, *Machine Intelligence*, volume 10, pages 455–470. Ellis Horwood, Chichester, 1982.

[CPQC03a] V. S. Costa, D. Page, M. Qazi, and J. Cussens. CLP(BN): Constraint logic programming for probabilistic knowledge. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 517–524, 2003.

[CPQC03b] Vitor Santos Costa, David Page, Maleeha Qazi, and James Cussens. Clp(bn): Constraint logic programming for probabilistic knowledge. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 517–52, San Francisco, CA, 2003. Morgan Kaufmann.

[CR00] C. Cumby and D. Roth. Relational representations that facilitate learning. In Anthony G. Cohn, Fausto Giunchiglia, and Bart Selman, editors, *KR2000: Principles of Knowledge Representation and Reasoning*, pages 425–434, San Francisco, 2000. Morgan Kaufmann.

[Cus99] James Cussens. Loglinear models for first-order probabilistic reasoning. In Kathryn Blackmond Laskey and Henri Prade, editors, *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI'99)*, pages 126–133. Morgan Kaufmann, 1999.

[Dec99] Rina Dechter. Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1-2):41–85, 1999.

[DFG03] Michelangelo Diligenti, Paolo Frasconi, and Marco Gori. Hidden tree markov models for document image classification. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(4):519–523, 2003.

[DL93] Paul Dagum and Michael Luby. Approximating probabilistic inference in Bayesian belief networks is np-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[DL97] Paul Dagum and Michael Luby. An optimal approximation algorithm for bayesian inference. *Artificial Intelligence*, 93:1–27, 1997.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38, 1977.

[DPDPL97] Stephen Della Pietra, Vincent J. Della Pietra, and John D. Lafferty. Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393, 1997.

[DR03] Rina Dechter and Irina Rish. Mini-buckets: A general scheme for bounded inference. *J. ACM*, 50(2):107–153, 2003.

[dSBAR05] R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of IJCAI-05, 19th International Joint Conference on Artificial Intelligence*, 2005.

[dSBAR06] R. de Salvo Braz, E. Amir, and D. Roth. Mpe and partial inversion in lifted probabilistic variable elimination. In *Proceedings of the 21sth National Conference on Artificial Intelligence (AAAI)*, 2006.

[FBBR05] Daan Fierens, Hendrik Blockeel, Maurice Bruynooghe, and Jan Ramon. Logical bayesian networks and their relation to other probabilistic logical models. In *ILP*, pages 121–135, 2005.

[Fen80] J. E. Fenstad. The structure of probabilities defined on first-order languages. In University of California Press, editor, *Studies in Inductive Logic and Probabilities*, pages 251–262, 1980.

[FGKP99] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[FH94] Alan M. Frisch and Peter Haddawy. Anytime deduction for probabilistic logic. *Artificial Intelligence*, 69(1–2):93–122, 1994.

[FHM90] Ronald Fagin, Joseph Y. Halpern, and Nimrod Megiddo. A logic for reasoning about probabilities. *Information and Computation*, 87(1/2):78–128, 1990.

[FL99] Peter Flach and Nicolas Lachiche. 1BC: A first-order Bayesian classifier. In Sazo Dzeroski and Peter Flach, editors, *Proceedings of ILP99*, pages 92–103. Springer-Verlag, 1999.

[FZ98] Enrico Fagiuoli and Marco Zaffalon. 2u: an exact interval propagation algorithm for polytrees with binary variables. *Artificial Intelligence*, 106(1):77–107, 1998.

[Gai64] H. Gaifman. Concerning measures in first-order calculi. *Israel Journal of Mathematics*, 2:1–18, 1964.

[GC93] R. P. Goldman and E. Cherniak. A language for construction of belief networks. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(3):196–208, 1993.

[Get00] Lise Getoor. Learning probabilistic relational models with structural uncertainty. *Lecture Notes in Computer Science*, 1864:322–329, 2000.

[GFKP01] L. Getoor, N. Friedman, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In S. Džeroski and N. Lavrac, editors, *Relational Data Mining*, pages 307–335. Springer-Verlag, 2001.

[GK95] Sabine Glesner and Daphne Koller. Constructing flexible dynamic belief networks from first-order probalistic knowledge bases. In *Symbolic and Quantitative Approaches to Reasoning and Uncertainty*, pages 217–226, 1995.

[GS82] Haim Gaifman and Marc Snir. Probabilities over rich languages, testing and randomness. *Journal of Symbolic Logic*, 47(3):495–548, 1982.

[Had94] P. Haddawy. Generating bayesian networks from probability logic knowledge bases. In R. Lopez de Mantaras and D. Poole, editors, *Uncertainty In Artificial Intelligence 10 (UAI94)*, pages 262–269. Morgan Kaufmann, 1994.

[Hai84] T. Hailperin. Probabilistic logic. *Notre Dame Journal of Formal Logic*, 25(3):198–212, 1984.

[Hal90] J. Y. Halpern. An analysis of first-order logics of probability. In *Proceedings of IJCAI-89, 11th International Joint Conference on Artificial Intelligence*, pages 1375–1381, Detroit, US, 1990.

[HCM+00] David Heckerman, David Maxwell Chickering, Christopher Meek, Robert Rounthwaite, and Carl Myers Kadie. Dependency networks for inference, collaborative filtering, and data visualization. *Journal of Machine Learning Research*, 1:49–75, 2000.

[Hec86] David Heckerman. Probabilistic interpretation for MYCIN's certainty factors. In L. N. Kanal and J.F. Lemmer, editors, *Uncertainty in Artificial Intelligence*, pages 167–196. Kluwer Science Publishers, 1986.

[HP90] M. Horsch and D. Poole. A dynamic approach to probabilistic inference using bayesian networks. In *Proceedings of the 6th Conference of Uncertainty in Artificial Intelligence*, pages 155–161. Morgan Kaufmann, 1990.

[HSC89] Eric Horvitz, Henri Jacques Suermondt, and Gregory F. Cooper. Bounded conditioning: Flexible inference for decisions under scarce resources. In *Proceedings of Conference on Uncertainty in Artificial Intelligence*, pages 182–193, Mountain View, CA, 1989. Association for Uncertainty in Artificial Intelligence.

[Jae97] M. Jaeger. Relational Bayesian networks. In Morgan Kaufmann, editor, *Proceedings of the 13th Conference on Uncertainty in Artificial Intelligence*, pages 266–273, 1997.

[JGJS99] Michael I. Jordan, Zoubin Ghahramani, Tommi Jaakkola, and Lawrence K. Saul. An introduction to variational methods for graphical models. *Machine Learning*, 37(2):183–233, 1999.

[JL87] J. Jaffar and J.-L. Lassez. Constraint logic programming. In *POPL '87: Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 111–119, New York, NY, USA, 1987. ACM Press.

[KDR00] K. Kersting and L. De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.

[KDR01] K. Kersting and Luc De Raedt. Towards combining inductive logic programming with Bayesian networks. In Celine Rouveirol and Michele Sebag, editors, *Proceedings of ILP 2001*, volume 2157 of LNAI, pages 104–117. Springer-Verlag, 2001.

[KH96] D. Koller and J. Y. Halpern. Irrelevance and conditioning in first-order probabilistic logic. In Howard Shrobe and Ted Senator, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Vol. 2*, pages 569–576, Menlo Park, California, 1996. AAAI Press.

[Kja93] Uffe Kjaerulff. *Aspects of Efficiency Improvement in Bayesian Networks*. PhD thesis, Aalborg University, 1993.

[KL88] M. Kifer and A. Li. On the semantics of rule-based expert systems with uncertainty. In *Lecture notes in computer science on ICDT '88*, pages 102–117, New York, NY, USA, 1988. Springer-Verlag New York, Inc.

[KLP97] D. Koller, Alon Y. Levy, and A. Pfeffer. P-CLASSIC: A tractable probablistic description logic. In *AAAI/IAAI*, pages 390–397, 1997.

[KP97] Daphne Koller and Avi Pfeffer. Learning probabilities for noisy first-order rules. In *IJCAI*, pages 1316–1323, 1997.

[KP98] D. Koller and A. Pfeffer. Probabilistic frame-based systems. In *Proceedings of the 15th National Conference on Artificial Intelligence (AAAI)*, pages 580–587, 1998.

[Lak94] Laks V. S. Lakshmanan. An epistemic foundation for logic programming with uncertainty. In *Foundations of Software Technology and Theoretical Computer Science*, pages 89–100, 1994.

[Lar03] David Larkin. Approximate decomposition: A method for bounding and estimating probabilistic and deterministic queries. In *Proceedings of the 19th Annual Conference on Uncertainty in Artificial Intelligence (UAI-03)*, pages 346–35, San Francisco, CA, 2003. Morgan Kaufmann.

[Las05] K. B. Laskey. First-order Bayesian logic. Technical report, George Mason University Department of Systems Engineering and Operations Research, 2005.

[LB87] Hector J. Levesque and Ronald J. Brachman. Expressiveness and tractability in knowledge representation and reasoning. *Computational Intelligence*, 3:78–93, 1987.

[LD93] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Routledge, New York, NY, 10001, 1993.

[LF03] N. Lachiche and P. A. Flach. 1BC2: a true first-order Bayesian classifier. In S. Matwin and C. Sammut, editors, *ILP02*, volume 2583 of *LNAI*, pages 133–148. Springer-Verlag, 2003.

[LK03] Martijn Leisink and Bert Kappen. Bound propagation. *Journal of Artificial Intelligence Research*, 19:139–154, 2003.

[LS94] Laks V. S. Lakshmanan and Fereidoon Sadri. Probabilistic deductive databases. In *Symposium on Logic Programming*, pages 254–268, 1994.

[Luc01] P. Lucas. Certainty-factor-like structures in bayesian belief networks. *Knowledge-Based Systems*, 14:327–325, 2001.

[Luk98] Thomas Lukasiewicz. Probabilistic logic programming. In *European Conference on Artificial Intelligence*, pages 388–392, 1998.

[Luk99] Thomas Lukasiewicz. Probabilistic deduction with conditional constraints over basic events. *J. Artif. Intell. Res. (JAIR)*, 10:199–241, 1999.

[LW96] Chao-Lin Liu and Michael P. Wellman. On state-space abstraction for anytime evaluation of bayesian networks. *SIGART Bull.*, 7(2):50–57, 1996.

[LY90] K. Lari and S.J. Young. The estimation of stochastic context-free grammars using the inside-outside algorithm. *Computer Speech and Language*, 4:35–56, 1990.

[Min95] Marvin Minsky. A framework for representing knowledge. In *Computation & intelligence: collected readings*, pages 163–189. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1995.

[Mit97] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill Higher Education, 1997.

[Mjo04] Eric Mjolsness. Labeled graph notations for graphical models: Extended report. Technical report, University of California Irvine – Information and Computer Sciences, 2004.

[MMR+05] Brian Milch, Bhaskara Marthi, Stuart Russell, David Sontag, Daniel L. Ong, and Andrey Kolobov. BLOG: Probabilistic models with unknown objects. In *Proc. IJCAI*, 2005.

[Mug95a] S. Muggleton. Inverse entailment and Progol. *New Generation Computing, Special issue on Inductive Logic Programming*, 13(3-4):245–286, 1995.

[Mug95b] S. Muggleton. Stochastic logic programs. In L. De Raedt, editor, *Proceedings of the 5th International Workshop on Inductive Logic Programming*, page 29. Department of Computer Science, Katholieke Universiteit Leuven, 1995.

[Mur02a] Kevin Patrick Murphy. *Dynamic bayesian networks: representation, inference and learning*. PhD thesis, University of California Berkeley, 2002. Chair-Stuart Russell.

[Mur02b] Kevin Patrick Murphy. *Dynamic Bayesian networks: representation, inference and learning*. PhD thesis, University of Berkeley, 2002. Chair-Stuart Russell.

[Nev06] J. Neville. *Statistical models and analysis techniques for learning in relational data*. PhD thesis, University of Massachusetts Amherst, 2006.

[NH95] L. Ngo and P. Haddawy. Probabilistic logic programming and Bayesian networks. In *Asian Computing Science Conference*, pages 286–300, 1995.

[Nil86] Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–88, 1986.

[NJG03] Jennifer Neville, David Jensen, and Brian Gallagher. Simple estimators for relational bayesian classifiers. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining*, page 609, Washington, DC, USA, 2003. IEEE Computer Society.

[NS92] R. T. Ng and V. S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[Pea88] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo (Calif.), 1988.

[Pea00] J. Pearl. *Causality: Models, Reasoning and Inference*. Cambridge University Press, 2000.

[Poo93] D. Poole. Probabilistic Horn abduction and Bayesian networks. *Artificial Intelligence*, 64(1):81–129, 1993.

[Poo96] David Poole. Probabilistic conflicts in a search algorithm for estimating posterior probabilities in Bayesian networks. *Artificial Intelligence*, 88(1–2):69–100, 1996.

[Poo97] D. Poole. The independent choice logic for modelling multiple agents under uncertainty. *Artificial Intelligence*, 94(1-2):7–56, 1997.

[Poo98] David Poole. Context-specific approximation in probabilistic inference. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 447–454, 1998.

[Poo03] D. Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, pages 985–991, 2003.

[Qui90] J.R. Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[Rab90] Lawrence R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. In *Readings in speech recognition*, pages 267–296. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[RD97] Luc De Raedt and Luc Dehaspe. Clausal discovery. *Machine Learning*, 26(2-3):99–146, 1997.

[RD04] M. Richardson and P. Domingos. Markov logic networks. Technical report, Department of Computer Science, University of Washington, 2004.

[Rie97] S. Riezler. Probabilistic constraint logic programming, 1997.

[RK04] Luc De Raedt and Kristian Kersting. Probabilistic inductive logic programming. In *ALT*, pages 19–36, 2004.

[RN03] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[Rot96] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, Apr 1996.

[Sha83] Ehud Shapiro. Logic programs with uncertainties: A tool for implementing expert systems. In *Proc. IJCAI'83*, pages 529–532. William Kaufmann, 1983.

[Sho75] Edward Hance Shortliffe. *Mycin: a rule-based computer program for advising physicians regarding antimicrobial therapy selection.* PhD thesis, Stanford University, 1975.

[SK97] Taisuke Sato and Yoshitaka Kameya. Prism: A language for symbolic-statistical modeling. In *IJCAI*, pages 1330–1339, 1997.

[SK00] T. Sato and Y. Kameya. A viterbi-like algorithm and em learning for statistical abduction, 2000.

[STBG94] D. Spiegelhalter, A. Thomas, N. Best, and W. Gilks. Bugs: Bayesian inference using gibbs sampling, version 0.30. Technical report, MRC Biostatistics Unit, University of Cambridge., 1994.

[TAK02] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proc. Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI)*, Edmonton, Canada, 2002.

[WBG92] M. P. Wellman, J. S. Breese, and R. P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, 7:35–53, 1992.

[Wüt95] Beat Wüthrich. Probabilistic knowledge bases. *IEEE Trans. Knowl. Data Eng.*, 7(5):691–698, 1995.

[ZP94] N. L. Zhang and D. Poole. A simple approach to Bayesian network computations. In *Proceedings of the Tenth Biennial Canadian Artificial Intelligence Conference*, 1994.

Author's Biography

Rodrigo de Salvo Braz was born in São Paulo, Brazil, on January 25, 1971. He graduated from Universidade de São Paulo in 1993 with a Bachelor of Science in Computer Science. He also obtained a Master of Science degree in Computer Science from the same department in 1998, while working for companies such as Bull Systems and PC Magazine Brazil. He spent two years as a graduate student at the Department of Cognitive and Linguistic Sciences at Brown University from 1998 to 2000. During his study in the Computer Science department at the University of Illinois, he focused his research on First-Order Probabilistic Inference and Natural Language Processing, and has published academic papers in highly selective conferences. Following the completion of his Ph.D., Rodrigo will continue his research career as a Postdoctoral Researcher at the Computer Science Division of EECS at University of California Berkeley, under the supervision of Prof. Stuart Russell.